Dissertation

Virtual Reconstruction of Hand-Torn Documents using Discriminative Models

Fabian Richter

Department of Computer Science

University of Augsburg

Advisor:	Prof. Dr. Rainer Lienhart
Reviewers:	Prof. Dr. Rainer Lienhart
	Prof. Dr. Bernhard Möller
	Prof. Dr. Eckehard Steinbach
Thesis defense:	November 16, 2015

Abstract

In numerous fields of computer vision such as in object detection, human pose estimation and image classification, machine learning has become an indispensable component for solving application-specific tasks. This thesis proposes and explores new ways of utilizing discriminative models for the virtual reconstruction of hand-torn documents.

In this work, reassembling pieces into document pages is accomplished in a bottom-up fashion. We show that discriminative models are suitable to solve various key problems and discuss how they can be fused effectively into a graph-based algorithm. In essence, we use our models to infer different spatial configurations between pieces, which are encoded into the graph's link structure. In contrast to the widely spread heuristic solutions, supervised learning has a solid theoretical foundation and thus enables a rigorous in-depth analysis of all key components of our proposed method.

We further investigate and thoroughly evaluate new methods for the representation of digital pieces. In order to deal properly with arbitrarily shaped pieces, we present a novel technique for the extraction of content-based features along their outer boundary. Our method allows an effortless integration of widely used features and therefore enables a highly discriminative, multimodal representation. We further propose a new color coding scheme based on the Fisher vector, which is extremely robust in the presence of noise and thus is ideally suited for real-world applications.

Besides, we introduce two novel, fully annotated datasets. In order to obtain a ground truth, human experts were asked to reassemble all digitized pieces into pages. This not only lays the basis for supervised learning from annotated examples but also provides the means for a rigorous evaluation. Inspired by existing benchmarks in the aforementioned domains we introduce two novel performance measures that quantitatively assess the quality of reconstruction results. We extensively evaluate our proposed method and demonstrate its general applicability on three different datasets, where we achieve state-of-the-art results.

Danksagung

Ich bedanke mich herzlich bei Prof. Dr. Rainer Lienhart für seine Betreuung und fortwährende Ermutigung in den vergangenen Jahren, insbesondere aber dafür, dass er mir beim Anfertigen meiner Doktorarbeit viele Freiräume eingeräumt hat. Ebenso bedanke ich mich vielmals bei Prof. Dr. Bernhard Möller für die Erstellung des Zweitgutachtens, sowie bei Prof. Dr. Eckehard Steinbach für die Anfertigung des externen Gutachtens.

Besonderer Dank gilt meinen Kollegen Christian X. Ries, Stefan Romberg, Johannes Schels, Christian Eggert, Dan Zecha und Christoph Lassner, die mit Ideen und zahlreichen konstruktiven Diskussionen direkt und indirekt zum Gelingen meiner Arbeit beigetragen haben. Besonders hervorheben möchte ich dabei Stefan Romberg und Christian X. Ries, ohne deren Unterstützung und Mitwirken meine Arbeit in dieser Form nicht zustande gekommen wäre. Stefan Romberg schulde ich auch insbesondere großen Dank für sein Engagement beim Korrekturlesen und die damit einhergegangenen wertvollen Anregungen. Dank schulde ich außerdem Jan Schneider und Dr. Bertram Nickolay vom Fraunhofer IPK für Ihre Anmerkungen zu meiner Arbeit.

Besonderer Dank gilt auch meinen Eltern Gabriele Richter und Wolfgang Richter, die mich stets in meinen Entscheidungen bestärkt und unterstützt haben. Außerdem möchte ich mich ganz herzlich bei all meinen Freunden bedanken, die mir immer eine moralische Stütze waren und ohne die ich mir die vergangenen Jahre nicht hätte vorstellen können. Ich danke außerdem meiner Partnerin Berit Eck, deren Verständnis und Unterstützung mir während des Schreibens in den vergangenen Monaten viel bedeutet hat.

Contents

Glossary

Terms

Document sheet	A sheet of paper with a front and a back side. The digitized version of each side is called a document *page* (see page 16).
Fragment	A physical piece which is obtained by tearing apart a document *sheet* by hand (see page 16).
Piece	Digitized version of a *fragment* showing either its front or its back side (see page 16).
Page	A page is the digital version of one of the two sides of a document *sheet*. Analogously to a sheet being composed of fragments, a page is composed of pieces (see page 16).
Counterparts	Two *pieces* stemming from the same *fragment*, i.e., the front and the back side of a hand-torn piece of paper (see page 158).
Support point	A point on the observed outer contour of a given piece that is part of the piece's polygonal approximation (see page 19).
Matching candidate	A pair of indices (i, j) representing the i-th support point on a piece $\mathcal{P}_k$ and the j-th one on $\mathcal{P}_l$, respectively (see page 50).
Inlier	A *matching candidate* whose *support points* are immediate neighbors in the ground truth, i.e., in the manually reconstructed *page* (see page 25).
Outlier	Any pair of *support points* that is not an *inlier* (see page 25).

Symbols

$\mathcal{P}_k$	Representation of a *piece* (see page 18).
S_k	Coordinates of observed contour points in either matrix or set notation. In the former case, each column represents the homogeneous coordinates of one point (see page 18).
$\hat{S}_k$	Coordinates of the *support points* defining a piece's polygonal approximation, in either matrix or set notation, same as for S_k (see page 19).
$H_{k,l,i,j}(\omega)$	A rigid transformation used for aligning two *pieces* $\mathcal{P}_k$ and $\mathcal{P}_l$ based on a single *matching candidate* (i, j) (see page 21).
Z_k	Rigid transformation that positions a digital piece via $Z_k\langle\mathcal{P}_k\rangle$ (see page 22).
G_k	Rigid transformation from ground truth that positions *piece* $\mathcal{P}_k$ as in the manually reconstructed *page* (see page 24).
$\hat{\mathbf{s}}_i^k$	Homogeneous coordinates of the i-th *support point* on *piece* $\mathcal{P}_k$, usually denoted as column-vector (see page 19).
C_{ij}	A set of inlier candidates for a given pair of anchor points (i, j) (see page 102).
$\mathbf{\Psi}_{k,l}(i, j)$	Multimodal feature vector representing the dissimilarity of *matching candidate* (i, j) between two *pieces* $\mathcal{P}_k$ and $\mathcal{P}_l$ (see page 29).

Functions

$\varsigma(Z_k, Z_l)$	Residual parameters of a transformation that corrects the position of *pieces* $\mathcal{P}_k$ and $\mathcal{P}_l$ as specified in the ground truth (see page 75).
$\Gamma_{k,l}(h, h')$	The *min-overlap* is a measure for the similarity of two hypotheses h and h'. Each hypothesis entails a spatial configuration of pieces $\mathcal{P}_k$ and $\mathcal{P}_l$ (see page 118).
$\Lambda_{k,l}(h^*)$	The *max-connectivity* of two *pieces* $\mathcal{P}_k$ and $\mathcal{P}_l$, assessed based on hypothesis h^* (ground truth) (see page 126).

Chapter 1

Introduction

"The better we understand dictatorship, the better we can shape democracy."
– Roland Jahn, *Federal Commissioner for the Records of the State Security Service of the former GDR*

1.1 Motivation

This thesis investigates the virtual reconstruction of hand-torn documents.

Human history is full of examples of criminals and suspects who try to eradicate their trails by tearing documents into pieces. An individual's motivation for doing so may for instance include tax fraud, business crime, or other criminal intentions. However, it is not only individuals who have an interest in making data inaccessible. A very prominent example dates back to East Germany and its Ministry of State Security, which is more commonly known as the *Stasi*. The Stasi was the former secret police of the German Democratic Republic (GDR). One of its main tasks was to collect sensitive data by spying on the population. Until 1989, the Stasi piled millions of surveillance files. In that year, the Peaceful Revolution, with its protest and violence-free demonstrations, marked the end of the Socialist regime. Along with the protests, citizens raided the Stasi headquarters in Berlin, where employees of the Stasi had begun to destroy the vast amount of records and documents they held. While some of the files were consigned to paper shredders or even set on fire, many of those files were simply torn into pieces by hand due to the lack of time. After the German reunification in 1990, those files were laid open to the public, so that citizens could inspect on their personal records. For the share of documents that have been shredded by hand it is thus of great interest to the general public that files get reassembled to unveil information about the often illegitimate activities of Stasi employees.

This example is by no means an isolated incident in history. Other governments also maintain files from dark days, including those in Poland, Czech Republic, Chile, and South Africa[1]. Very often, the number of documents that need to be reconstructed is too large for a manual reconstruction to be feasible. In case of the Stasi files, investigators are faced with about 600 million pieces from an estimated 45 million documents that need to be reassembled. This overwhelming problem size makes clear that a computer-aided approach is needed to enable a semi-automatic virtual reconstruction. In 2007, the Fraunhofer IPK (Institute for Production Systems and Design Technology) started a pilot project for the virtual reconstruction of a small portion of the Stasi files. In the period from 2007 to 2014, the "ePuzzler", which is the reconstruction software that has been developed over the years, has led to a reconstruction of roughly $29{,}000$ pages[2]. Over the next couple of years, the last milestone of this ambitious project aims at the digitization of pieces from a total of 400 bags, as well as the reconstruction of these pieces into approximately 1 million pages. Research projects like this that facilitate a virtual reconstruction of documents are certainly of great interest to governments, ministries of finance, and public prosecutor's offices.

In the following section we briefly outline some of the most important challenges related to the virtual reconstruction of hand-torn documents. Besides, we also discuss related problems and potential applications.

1.2 Related Problems and Applications

The problem of reconstructing hand-torn documents can be regarded as a special case of the more general setting of having to reconstruct fragmented 2D objects. Looked at the problem in this light, one could generally think of adapting our approach to solve similar tasks, for instance, in related research domains like archaeology, art restoration, or forensics.

Real-World Examples

One intriguing related problem is that of dealing with historic documents. For example, the Historical Archive of the City of Cologne was almost completely destroyed in 2009 when the ground beneath it collapsed, causing a large amount of valuable historical documents to be buried under debris. Certainly, some of the documents got destroyed entirely, yet some of them have not been damaged beyond reconstruction,

[1] https://www.heise.de/artikel-archiv/ct/2014/18/076_Das-Riesenpuzzle, for the printed article see c't 2014, issue 18, p. 76–81.

[2] http://www.faz.net/-gpg-7rzf9

as a feasibility study conducted by the Fraunhofer IPK has shown. Another example is that of an Israeli group who recently scanned 250,000 historic document fragments that are hundreds of years old[1]. In both of these examples, one has to account for the fact that those fragment may have been exposed to weathering processes and natural decay. In such an instance one not only needs to take special care during digitization to prevent further destruction, but also the reconstruction system itself needs to account for physical material loss along the pieces' boundaries.

The problem of reconstructing documents from paper shredder snippets is also related to our problem. However, fragments from paper shredders differ from hand-torn document fragments in several aspects: First, while documents most often are torn by hand into only a very limited number of pieces (say, in the order of tens), commercial shredders typically tear apart a single sheet into hundreds or thousands of similarly shaped chads. In 2011, the U.S. Defense Advanced Research Projects Agency (DARPA) ran a Shredder Challenge[2], in which the goal was to unshred five documents in order to extract hidden messages written on them. In this challenge, instead of requiring a complete reconstruction of pages, the main aim was to extract information from the pages in the shortest time possible. According to Norman Whitaker, deputy director of DARPA's Information Office, the problem of document reconstruction is "just one facet of information assurance, an aspect that is often overlooked" [24]. Apart from the overwhelming number of pieces per document, one also has to deal with the fact that the shredded snippets convey little to no shape information. This stands in bold contrast to hand-torn documents, where fragments quite often feature very distinct outer contours. Due to the size and complexity of the problem, it comes at no big surprise that the winning approach of the DARPA Shredder Challenge was not fully automatic. The team partly relied on automatically generated piece-pair recommendations that then had to be reviewed by humans for final decision making.

Tiling Puzzles

Apart from the abovementioned real-world examples, there are many artificial puzzles related to our problem. We give a brief summary of two types of puzzles belonging to the category of *tiling puzzles*, which are discussed thoroughly in [13]. The scope of a tiling puzzle is to pack pieces into a predefined shape without producing any overlap or gaps in between them.

Jigsaw puzzles are among the most popular type of tiling puzzle, which tradition-

[1] http://spectrum.ieee.org/podcast/computing/software/a-digital-jigsaw-puzzle
[2] http://archive.darpa.mil/shredderchallenge/

ally were made from wood and commonly used as educational toys. One important characteristic of jigsaw puzzles is that they typically have a global image that guides the puzzler. Early works in computer science focused on digitizing puzzle pieces in order to solve the problem algorithmically. Nowadays, however, a more widely studied variant in the computer vision community is that of solving purely digitial, *square jigsaw puzzles*. In this form of puzzle, one dissects a digital natural image into square tiles. Since all pieces have identical shape and the frame of the puzzle is known, finding the solution comes down to positioning pieces correctly on a regular grid. The most widely studied twists to this definition assume that – beside the unknown location – the *orientation* of pieces is also unknown. Compared to a puzzle with N pieces having fixed orientation, the number of possible solutions increases multiplicatively by a factor of 4^N. This not only makes the puzzle more challenging from a combinatorial perspective, one also needs to account for rotations of pieces algorithmically. A more comprehensive discussion of three common variations and possible solutions of the square jigsaw problem is given in [23].

A closely related type of tiling puzzle is called *edge-matching puzzle* [27]. An edge-matching puzzle involves identically shaped pieces, which need to be arranged such that edges between adjacent tiles have the same "pattern". Typically, tiles are simply squares, and each of the pieces' four edges is colored one of several colors. That is, for two adjacent tiles to match, both pieces must be identically colored. As a consequence, even if two pieces fit together regarding their pattern, one still does not know whether or not those pieces make for a valid partial solution; only once the puzzle is completed one can assess the correctness of local parts of the solution. In contrast to square jigsaw pieces "cut out" from a digital image, edge-matching puzzles offer no global guiding image and hence are difficult to solve for humans. Due to the inherent ambiguity in matching pieces, finding a solution can be very computationally intensive. In fact, as shown in [13], this type of puzzle is NP-complete. A very prominent example of edge-matching puzzles is the *Eternity II puzzle*. It is a competition released by Christopher Monckton in 2007, who offered a price money of $2 million to the first person finding a complete solution. In spite of the fact that this puzzle consists of only 256 square pieces, neither hobby puzzlers nor computer scientists could find a solution since then. The competition ended officially in December 2010 with the price money being left unclaimed.

Comparison between Artificial Puzzles and Real-World Problems

In contrast to artificial puzzles, having to deal with real-world pieces leads to several

practical problems: First of all, an inevitable consequence of tearing documents into pieces is that, due to small paper fibers along the boundary, the physical contour is often ill-defined. Since the tearing proceeds along the surface as well as the thickness of the paper, matching edges can partly *overlap* each other. On the other hand, material loss may cause adjacent pieces to have small *gaps* in between them. Moreover, since physical fragments need to be scanned, their digitized equivalents may contain *digital imperfections*. For instance, pixel colors along matching edges may vary slightly due to shadowing or image quantization effects, which complicates the identification of matching pieces.

Another very important aspect is that document pieces have arbitrary shape and for this reason provide *no well-defined sides*. This makes the matching of pieces conceptually more difficult because pixel-correspondences across the tearing boundary are not known. This is exacerbated by the fact that paper fragments are scanned under arbitrary rigid transformations. That is, we generally have no a priori knowledge about the pieces' *orientation* and thus two pieces can be arranged adjacently in virtually infinitely many ways.

1.3 Contributions

This thesis makes the following main contributions:

- **Piece representation:** Due to the arbitrary shape of document pieces, obtaining a content-based feature representation is nontrivial. To alleviate this problem, we propose a novel technique called the *local coordinate embedding*, which extracts a straightened pixel band along the contour. This allows an effortless integration of different widely used features, which in turn enables a multimodal representation of pieces. For this representation we also introduce a new color encoding scheme based on the Fisher vector, which gives superior performance on both synthetically generated as well as real-world pieces.
- **Reconstruction algorithms:** We develop two adaptations of Kruskal's spanning tree algorithm for the simultaneous reconstruction of multiple document pages. Based on our multimodal representation of pieces composed of geometric and content-based features, both approaches employ supervised learning to define the underlying graph's link structure. The *key distinction* between our two algorithms lies in the utilization of supervised learning for different purposes:

  (i) Our first approach employs adaptive boosting to identify contour points across two pieces that were putatively adjacent in the original document. Since

this binary classification approach only considers two points at a time, we additionally encode point-pairs into geometric signatures for further postprocessing. We demonstrate how these verified point-pairs are used to determine multiple spatial configurations of each individual piece-pair. Our graph algorithm disambiguates these configurations step-by-step by merging piece-pairs in a bottom-up manner. To recover the correct spatial arrangement of pieces, we update the graph's link structure after each merging step, allowing the incorporation of additional evidence gathered throughout the reconstruction.

(ii) Our second reconstruction approach is heavily tuned towards efficiency. To this end, we revisited the previous pipeline and developed a new two-step procedure for the identification and ranking of spatial configurations of piece-pairs. For the first step we introduce a novel, highly efficient variant of the M-estimator SAmple Consensus (MSAC) method, which is tailored specifically to the alignment of piece-pairs. Our approach exploits information from the pieces' outer contours as well as their foreground regions to quickly discard invalid spatial configurations. In the second step we make use of a discriminatively trained cost model. The purpose of this model is to rank all configurations according to the compatibility of the pieces' adjacent boundary regions. Instead of relying on a geometric verification of binary classification results, we show that *structured output prediction* is directly applicable to model the interrelation among *sequences* of point-pairs.

- **Human-annotated datasets:** In contrast to most other research domains in the fields of computer vision and machine learning, there is a lack of publicly available benchmark datasets for the evaluation of reconstructed documents. For this reason we introduce two novel, fully annotated datasets comprising document pages with different degrees of fragmentation: Each paper sheet has been torn by hand into 8, 16, 24 and 32 fragments. To enable an objective quantitative evaluation based on a ground truth, digitized pieces have been reassembled virtually by human experts.
- **Quantitative performance evaluation:** Throughout the thesis we introduce two novel performance measures that enable a rigorous quantitative evaluation of reconstruction results:

  (i) We introduce the notion of *adjustment cost*, which intuitively quantifies the degree of misalignment between pieces. Our idea was that inaccurately reconstructed document pages should naturally entail high adjustment costs, whereas near-perfect solutions should induce low costs.

(ii) Our second performance measure is the *average precision*, a standard technique used for the evaluation of object detection and image retrieval systems. We demonstrate how this performance measure can be adopted to our problem domain and discuss its use in different evaluation scenarios of high practical relevance.

1.4 Thesis Overview

The thesis presents two different yet conceptually related approaches to the virtual reconstruction of hand-torn documents. Our approaches were developed in consecutive steps for the most part: We started with a straightforward approach based on binary classification. On the basis of our findings we then devised a structured prediction approach that overcomes the observed shortcomings. This process is reflected by the structure of this thesis, which is organized in four parts:

The first part starts with a short review of related work in chapter 2. It then introduces the datasets used throughout this work and discusses fundamental techniques related to feature extraction. In chapter 3 we give details of our image acquisition and data preprocessing. Afterwards, we discuss the notation and explain how document pieces can be repositioned through rigid transformations. In that context we also introduce a representation for groups of aligned pieces. The chapter is concluded with a discussion of the ground truth, which lays the foundation for a thorough quantitative evaluation. In chapter 4 we then discuss different geometric and content-based features that will be commonly used in the remainder of this work.

In the second part we present a first approach to solve the "restricted" document reconstruction problem. Here we make the simplifying assumption that the belonging of pieces to document pages is known in advance. In chapter 5 we introduce a binary classification approach with additional subsequent spatial verification through geometric signatures. The resulting verified point-correspondences form the basis for the hill-climbing optimization discussed in chapter 6, which is used to determine the optimal spatial configuration of piece-pairs. In chapter 7 we present a spanning tree algorithm for the reconstruction of document pages. The purpose of this graph-based algorithm is to recover the layout of individual document pages in a bottom-up fashion. Finally, we introduce a novel performance measure to which we refer as the mean Adjustment Cost (mAC). The mAC asseses the quality of a given reconstructed page with respect to the ground truth and therefore enables an objective quantitative evaluation.

In the third part we revisit our first approach to come up with an equally effective yet more efficient approach. The alignment of piece-pairs is now approached in chapter 8 by performing partial contour matching. For this purpose we introduce a novel, highly optimized variant of the M-estimator SAmple Consensus (MSAC) method. Afterwards, we formulate the problem of aligning and ranking piece-pairs as a structured output prediction problem in chapter 9. We demonstrate how a structural support vector machine can be adopted to the task of assessing the pieces' compatibility regarding geometric and content-based features. Using supervised learning on the basis of *piece-pairs* is a more holistic approach as compared to applying binary classification to individual *point-pairs*. This fact is reflected by our revised reconstruction algorithm, which is conceptually the same as the one presented in the second part but is more straightforward. We discuss this matter in chapter 10, where we also evaluate the performance of our new approach in the "unrestricted" reconstruction scenario: We give up on knowledge about the belonging of pieces to pages and show that we are capable of reconstructing multiple document pages simultaneously. To quantify the quality of reconstruction results, we use the average precision (AP) as our performance measure, which is widely recognized as the de facto standard in other computer vision fields.

Finally, the last part concludes the thesis with a brief discussion.

Part I

Foundations and Related Work

Chapter 2

Related Work

The automated solving of jigsaw puzzles is a traditional computer vision problem that has been studied extensively in many variants, going back to as early as the work of Freeman and Garder [18] in 1964. While many early problem definitions included the identification and matching of shapes [7, 18, 25, 60, 64], research in the last years more often focuses on matching image content [2, 11, 23, 40, 48, 62]. These recent approaches use images that have been digitally cut into many square pieces, and hence the pieces provide no shape information. Nowadays it is possible to solve instances of the square jigsaw puzzle in the order of 10^3 pieces per image. This is for instance accomplished by Gallagher [23], who proposes a gradient-based compatibility measure and utilizes Markov Random Fields to find the optimal solution. Another recent approach is that of Sholomon et al. [48], in which the authors adapt a genetic algorithm to solve the puzzle. By introducing a problem-specific crossover operator they achieve highly accurate reconstruction results for puzzles containing up to $22{,}834$ square pieces.

Real-world problems that are more closely related to the reconstruction of hand-torn documents include applications in archaeology [12, 36, 38] and forensics [49]. In contrast to the aforementioned jigsaw puzzles, pieces in these applications have arbitrary shape and thus provide no unambiguously defined sides. This lack of predefined sides naturally complicates the matching of pieces. Especially for archaeological findings, where objects are often damaged and edges are sanded down by erosion, outer boundaries and corner points provide no robust information. This kind of material loss makes an accurate reconstruction even more difficult. Although the contours of hand-torn documents are sometimes also corrupted by noise, they can obviously be considered to be more reliable than those of fragmented objects that were exposed to weathering processes.

Regarding the representation of the fragments' outer boundaries, most approaches in the literature can be categorized into two approaches: Either the outer boundaries are (i) sampled uniformly, such as in [3, 12, 51], or are (ii) described by critical points such as through corner points or other polygonal approximations [5, 8, 33, 34, 36, 38, 49, 66]. Once a sufficiently accurate approximation of boundaries is obtained, the next step of reconstruction usually involves a partial contour matching. One way of doing so is based on the turning function proposed by Wolfson [61]. The idea behind this is to encode turning angles along a polygon as a function of their arc length. Since two polygons with well-matching contour segments should have similar shapes, the corresponding parts of their turning functions also must be similar. The turning function is for instance used in [51, 66, 67] to form a string-based representation. In the latter work of Stieber et al. [51] the authors use this representation in conjunction with dynamic programming to find the best matching boundary segments between two pieces. The Smith-Waterman algorithm [50], which is used in this work for the substring matching, is also applied in [7, 10] to efficiently match other 2D objects.

At their core, document reconstruction approaches all rely on some kind of partial contour matching. Based on the matching results, pairs of aligned pieces are then hypothesized and ranked, before finally being merged into partial solutions. For example, in [12] the authors compare curvature-encoded fragment outlines by dynamic programming. Since their approach represents pieces at progressively increasing scales of resolution, the computational cost of finding optimal matches is reduced. This makes their method viable for problems of practical size in the order of thousands of fragments.

A different route is taken by Zhu et al. [66], who aim to find a globally consistent solution to the reconstruction task. After identifying initial candidate matches based on the turning function, the authors disambiguate these candidates by considering the spatial compatibility of neighboring pieces. Then, to obtain a consistent solution, they use an iterative procedure that alternates between gradient projection and merging steps.

Another reconstruction method that is closely related to our work is that of Cao et al. [8]. Their framework for the reconstruction of photos inspired several aspects of our work, such as the representation of digital pieces with different feature modalities as well as the use of a spanning tree algorithm for the actual reconstruction task. The key distinction to their work is that we (i) use a discriminatively trained cost model for partial contour matching and (ii) conduct extensive experiments to evaluate our approach quantitatively. A second closely related approach is that of Stieber et al. [51].

Their procedure for reassembling hand-torn documents is much related to our work for a number of reasons: First of all, to the best of our knowledge, it is the only other work that deals with the problem of reconstructing multiple document pages without knowing the belonging of pieces to pages. More importantly, however, the authors also provide quantitative results, which unfortunately is not common practice due to the scarcity of publicly available datasets. From an algorithmic perspective, their work is related to ours because the authors also apply the Smith-Waterman algorithm to align contour points between pairs of document pieces.

Chapter 3

Datasets

3.1 Introduction

In this chapter we introduce three real world datasets that are going to be used throughout this work. Although notable efforts have been made to approach the problem of document reconstruction, there is, to the best of our knowledge, no benchmark dataset publicly available. For this reason we decided to make our own dataset available online [1]. Besides having no common dataset, we found that the research community lacks a unified benchmark, which renders a fair comparison of approaches infeasible. Hence our ambition was not only to provide a novel, publicly available dataset, but also to introduce performance criteria for quantitative evaluation that provide meaningful insights and are easily comprehensible.

To enable quantitative evaluation, we created an annotation tool that allowed human experts to reassemble individual pages digitally. As we will see shortly, this not only gives us the means for an objective evaluation, but also, knowledge about the adjacency of piece-pairs enables the use of supervised learning methods from annotated examples.

In this chapter we describe our used datasets and fundamental concepts required for our reconstruction method. After an introduction about our datasets in section 3.2, we discuss our image acquisition and data preprocessing methodology in section 3.3. In section 3.4 we remark on notation commonly used throughout this thesis. Afterwards, section 3.5 explains the fundamentals of rigid transformations. These are subsequently used in sections 3.6 and 3.7 for the repositioning of individual pieces and the formation of groups of aligned pieces. Finally, in section 3.8 we discuss our an-

[1] http://www.multimedia-computing.de/wiki/An_Annotated_Dataset_of_Shredded_Documents

Figure 3.1: Example pages from the *bdw082010* dataset.

notation tool, which was used to obtain a ground truth from manually reconstructed pages. In this section we also discuss the fundamentals of how the annotations can be utilized to assess the quality of reconstruction results.

3.2 Datasets, Datasplits and Best Practice

The aim of our work is to develop a method capable of reassembling documents regardless of their textual and visual content. We use three different datasets in total, two of which have been created by ourselves.

For our first dataset we decided to use the scientific magazine "*Bild der Wissenschaft*" (issue 08/2010) because it features text pages, natural images, concept art, and various other content elements. We have deliberately chosen this magazine in order to avoid being biased towards only text pages or photos. A few example pages that illustrate the rich variety of image contents are depicted in figure 3.1.

In the following we refer to this dataset as *bdw082010*[1]. In total it consists of 48 document *sheets* whose front and back sides correspond to 96 *pages*. All sheets have been partitioned into three disjoint datasplits: {train} (16 sheets), {val} (8 sheets), and {test} (24 sheets). We were careful to ensure that these subsets are balanced regarding different contents (e.g. text only, concept art, natural images, and combinations). The dataset comes in four degrees of fragmentation of 8, 16, 24 and 32 pieces per page. Each paper sheet was first torn into 8 physical *fragments*. Afterwards, fragments have been scanned in order to obtain digital *pieces* (see section 3.3). Pages with a higher degree of fragmentation have been obtained by tearing sheets in a hierarchical manner: We have repeatedly split fragments further to also obtain sheets consisting of 16, 24, and 32 fragments, respectively. This first dataset is commonly used throughout the following chapters for training, parameter evaluation (on {val}), and testing.

[1] We thank the editorial staff of the *Bild der Wissenschaft* and the publisher *Konradin Medien GmbH* for their permission to use the magazine and to make the dataset publicly available for research.

	bdw082010					*booklet*		*stieber500p*
datasplits	{train}		{val}	{test}		{test}		{test}
pieces per page	8, 16	24, 32	8, 16	8, 16	24, 32	8, 16	24, 32	varies ($\varnothing = 8.33$)
number of pages	32	12	16	48	12	24	12	60

Table 3.1: Overview of the three datasets used in this work. The first two (*bdw082010* and *booklet*) come with document sheets that were progressively torn into up to 32 pieces. The three datasplits thus offer four different levels of fragmentation.

For our second dataset called *booklet* we used an information brochure. Pages from this brochure complement those of *bdw082010* in that they are printed on thicker paper. Therefore, fragments feature slightly different physical characteristics along their tearing boundaries. Due to the texture and the thickness of the paper, shearing effects are typically more pronounced, often leading to substantially more noise along the fragments' boundaries (see figure 3.3). Furthermore, pages in the *booklet* dataset show mostly homogeneous image contents and have a different aspect ratio as compared to those in our first dataset. More importantly, this dataset is strictly reserved for evaluation purposes and thus consists of only a {test} datasplit. The entire dataset consists of 12 sheets (24 pages) featuring the same degrees of fragmentation as for *bdw082010*.

In order to complement our own two datasets, we use a third party dataset introduced in [51]. We dubbed this dataset *stieber500p* according to the first author's name. It consists of 250 physical fragments stemming from 30 sheets, each of which features a different degree of fragmentation. On average, each sheet has been torn into 8 fragments.

The characteristics of all three datasets are summarized in table 3.1.

3.3 Image Acquisition and Digital Preprocessing

For digitization we used an off-the-shelf A4 scanner (Canon CanoScan 9000F), which only allows one to scan fragments from one side at a time. To facilitate the segmentation of the pieces' foreground, we equipped the scanner with a unicolor foil. We used a bright green cast foil[1], which has no texture and a plain, smooth surface. All built-in auto corrections of the scan software have been disabled to obtain a consistent digitization result. We found that 200 dpi are sufficient for our needs, and all fragments were scanned using this resolution. Before the scanning took place, all fragments were carefully separated from each other for an easier postprocessing. Examples for digi-

[1] The color was deliberately chosen as it is almost never present in magazines and hence provides a high contast background.

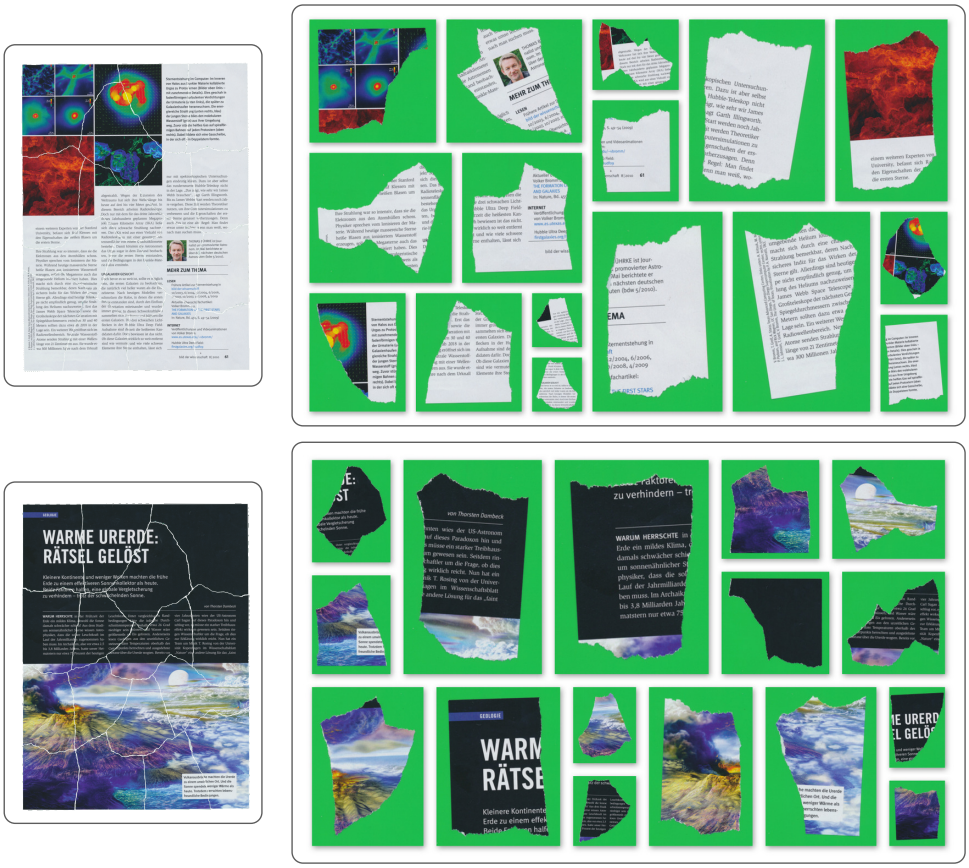


Figure 3.2: Examples for 32 digitized pieces from the *bdw082010* dataset (front and back side of a single sheet). **Left:** Manually reconstructed pages. **Right:** Scanned pieces of each page that have been scaled for a more compact visualization.

tized pieces from the *bdw082010* and the *booklet* dataset are shown in figure 3.2 and figure 3.3, respectively.

After digitization, we apply a threshold in HSV color space to the scan result in order to subtract the unicolored background from the pieces' foreground. As depicted in the figures, some pieces have foreground regions that have similar color as our scan background (green). To obtain a binary segmentation mask for each piece, we perform a morphological opening (an erosion followed by a dilation) on the thresholded image. This step results in a single connected component, one for each individual piece. A single *digital piece* is represented by $\mathcal{P}_k = (S_k, \hat{S}_k, I_k)$, which we obtain from the following preprocessing steps: First, the binary segmentation mask of a given piece is input to the algorithm of Suzuki et al. [52], which determines a set of outer contour pixels $\boldsymbol{s}_i^k \in S_k$. We refer to this set of points as the piece's *observed contour*.

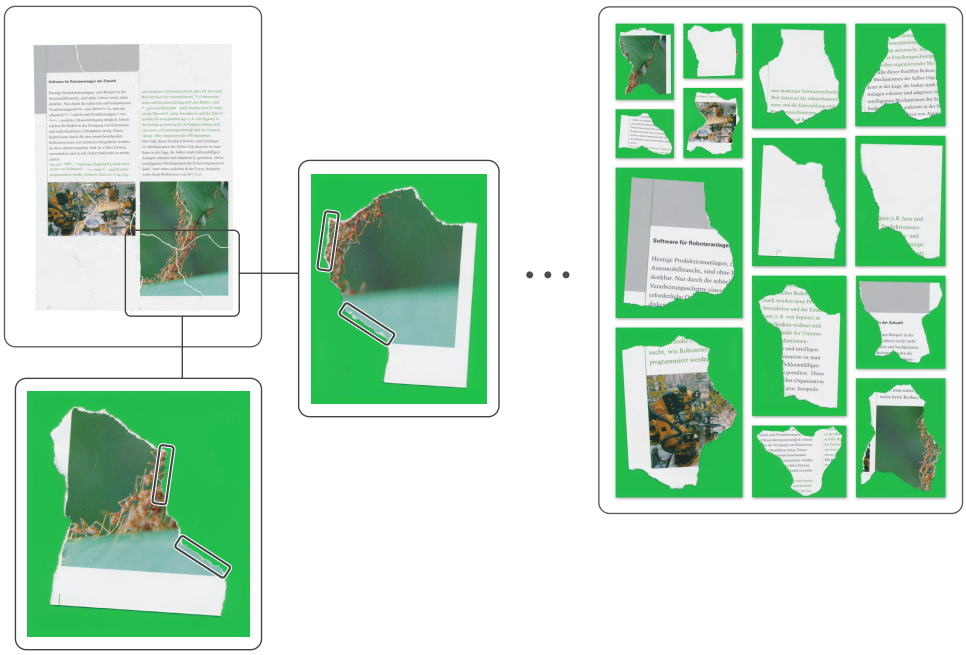


Figure 3.3: Example for a manually reconstructed page from the *booklet* dataset. Since the brochure was printed on thick paper, shearing effects along the pieces' contour are more pronounced as in *bdw082010* dataset. Observed contours on the scanned fragments differ from their ideal outlines, and small paper fibers and loss of tiny fragments lead to partly incompatible matching segments on adjacent pieces (see the highlighted segments).

Next we apply the Douglas-Peucker algorithm [14] to this set. The purpose of this algorithm is to reduce the number of points required to represent the piece's outer contour. Regarding a predefined threshold, it chooses $n(k)$ *support points* that represent a *closed polygonal curve* (or *polygon* in short). The number of points needed for a sufficiently accurate approximation depends on the piece's size and the complexity of the contour's shape. We represent the closed polygonal curve by its set of support points $\hat{S}_k = \left\{\hat{\boldsymbol{s}}_i^k\right\}_{i=1}^{n(k)} \subset S_k$.

This step is motivated from both a theoretical and practical perspective: Representing the boundary of a piece by a closed polygonal curve greatly reduces the number of contour points that need to be considered for the matching of pieces, thereby enabling a more efficient reconstruction. Throughout this thesis we assume that both contour pixels as well as support points are arranged on a cycle in counterclockwise direction. That is, points in S_k and $\hat{S}_k$ should be thought of as *cyclically ordered sets* [1]. Instead of their representation by sets we often interchangeably organize points in matrices.

[1] Formally, cyclically ordered sets require a ternary relation to define the orientation of the cycle (see chapter 8 for details).

Therefore, we use $\hat{S}_k = \left[\hat{\boldsymbol{s}}_1^k, \ldots, \hat{\boldsymbol{s}}_{n(k)}^k\right]$ as an alternative representation in form of a $3 \times n(k)$ matrix, in which we store homogeneous point coordinates as column vectors.

We obtain I_k from the scanned image by using its color values at all pixel locations identified through the segmentation mask. On the lefthand side in figure 3.2 we show two examples for manually reconstructed pages. The pieces' observed outer contours can best be seen on the righthand side, which depicts individual pieces on the scan background after digitization.

3.4 Notation and Conventions

Some words on the notation and conventions used throughout this thesis:

A lower-case letter in bold type denotes a column vector, which for instance is used to represent two-dimensional spatial coordinates of points. For convenience we most often use homogeneous coordinates, e.g., to represent coordinates (x, y) we would write $\boldsymbol{s} = (x, y, 1)^T$. Quantities with different semantics (e.g., a list of parameters of a function, or a combination of scalars and matrices) are written as n-tuple and denoted by a lower-case letter in regular type.

Functions are most often written in the form $f_q(x; p) = y$ where x is the input from some domain $\mathcal{X}$, p is a n-tuple representing the function parameters, $y \in \mathcal{Y}$ is the output, and q is another n-tuple to represent a list of *qualifiers*. We make use of qualifiers as a kind of parameters whose purpose is to disambiguate examples. For instance, $f_{q_1}(x; p)$ and $f_{q_2}(x; p)$ means that we evaluate function f with the same input x and identical parameters p, however, the outcome is computed from separate data qualified by q_1 and q_2, respectively.

We commonly use the unary operators $\lfloor x \rfloor =_{df} \max\{y \in \mathbb{Z} \,|\, y \leq x\}$ and $\lceil x \rceil =_{df} \min\{y \in \mathbb{Z} \,|\, y \geq x\}$, which are the floor and ceiling functions that map any real-valued number to the largest previous and smallest following integer, respectively. Similarly, binary operators $\lfloor x, y \rfloor =_{df} \min\{x, y\}$ and $\lceil x, y \rceil =_{df} \max\{x, y\}$ are used to determine the minimum and maximum of two values.

Angles are measured in counterclockwise direction in the right-handed Cartesian coordinate system, where the x-axis points to the right and the y-axis points upwards. To represent the pieces' image coordinates we use the left-handed orientation instead, for which the x-axis is left unchanged but the y-axis points downwards. This orientation of axes is the standard image convention for most computer vision applications, to which we refer as the *image coordinate system*.

3.5 Rigid Transformations

The relation between two pieces that need to be aligned is represented by a single rigid transformation matrix H. A rigid transformation performs an in-plane *rotation* as well as a *translation* and thus has 3 degrees of freedom (two components of translation and one angle of rotation). The purpose of H is to align one piece $\mathcal{P}_l$ with another piece $\mathcal{P}_k$ while holding piece $\mathcal{P}_k$ fixed in position. Accordingly, one can interpret H as a *hypothesis* for embedding the two pieces into a joint coordinate system.

Let $\mathcal{P}_k$ and $\mathcal{P}_l$ be the two pieces to be aligned, and let i and j index a support point each on either contour. Computing a rigid transformation for alignment actually involves a chain of transformations: First of all, we superimpose both support points into a rotation center by translating one piece while holding the other piece in position. Without loss of generality, let $\mathcal{P}_k$ be the piece that is held in position. By applying a translation matrix $T(\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_j^l)$ to $\mathcal{P}_l$ we shift that piece such that points i and j have the same coordinates thereafter. Finally, we rotate $\mathcal{P}_l$ around point $\hat{\boldsymbol{s}}_i^k$ by angle ω. Since we propose and evaluate different approaches for estimating the rotation angle throughout this thesis, the discussion on how to obtain ω is postponed to later sections.

Rotating about the superimposed point $\hat{\boldsymbol{s}}_i^k$ is accomplished in three steps: Shift the rotation center to the origin, perform an in-place rotation by ω, and finally back-shift the piece afterwards. The combined transformation $T(\hat{\boldsymbol{s}}_i^k)R(\omega)T(-\hat{\boldsymbol{s}}_i^k)$ can be written as a single homogeneous matrix. We write

$$T(\hat{\boldsymbol{s}}_i^k)R(\omega)T(-\hat{\boldsymbol{s}}_i^k) = \begin{pmatrix} \alpha & \beta & (1-\alpha)r_x - \beta r_y \\ -\beta & \alpha & \beta r_x + (1-\alpha)r_y \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.1}$$

with $\alpha = \cos(\omega)$, $\beta = \sin(\omega)$, and $\hat{\boldsymbol{s}}_i^k = (r_x, r_y, 1)$. Since the image coordinate system has its origin in the top-left corner (with the y-axis pointing downwards), positive values of ω mean counterclockwise rotation.

The hypothesis H that combines translation and rotation is obtained in closed form as the following sequence of 3×3 transformation matrices:

$$H_{k,l,i,j}(\omega) = T(\hat{\boldsymbol{s}}_i^k)R(\omega)T(-\hat{\boldsymbol{s}}_i^k)T(\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_j^l) \tag{3.2}$$

The subscripted (k, l, i, j) is a qualifier to make clear that piece $\mathcal{P}_l$ is aligned with $\mathcal{P}_k$, using the pieces' j-th and i-th support point, respectively. If it is clear from the context, we often drop qualifiers or parameter ω and write briefly $H = H(\omega) = H_{k,l,i,j}(\omega)$.

3.6 Applying Transformations to Pieces

Throughout the actual reconstruction, the position of pieces is updated in an iterative manner. For example, a single piece could first be aligned to another piece, which together form a partial solution. Afterwards, these two pieces may again be repositioned jointly by another rigid transformation. Consequently, the reconstruction typically involves multiple rigid transformations H being applied to each piece. The corresponding sequence of transformations is here subsumed into a single matrix $Z_k \in \mathbb{R}^3$, which is obtained by multiplying the homogeneous transformation matrices in the same order as they are applied to a given piece $\mathcal{P}_k$.

We now demonstrate how the digital piece has to be altered in order to account for its updated position. Let $\mathcal{P}_k = (S_k, \hat{S}_k, I_k)$ denote the piece after digitization. Recall also from section 3.3 that the first two components are point sets representing the observed and the approximated contour, and let I_k be again the image showing the piece's foreground. In order to update the representation of $\mathcal{P}_k$ regarding transformation Z_k, we define:

$$Z_k\langle\mathcal{P}_k\rangle =_{df} (Z_k S_k, Z_k \hat{S}_k, J_k) \tag{3.3}$$

Here we use the matrix representation of $\hat{S}_k$, which stores the $n(k)$ many support points as column vectors. Since points are represented in homogeneous coordinates, the matrix dimensions are $3 \times n(k)$. Thus, the matrix product $Z_k S_k$ denotes the points' coordinates after the update through Z_k. Likewise we proceed with the observed outer contour.

The updated image J_k in eq. (3.3) is obtained through geometrical image transformation with bilinear interpolation. Our implementation relies on the OpenCV library [1], which avoids sampling artifacts by proceeding in reverse order, from the destination to the source image. The library computes the inverse mapping $\langle f_x, f_y\rangle : J_k \rightarrow I_k$ based on the inverse transformation matrix Z_k^{-1}. For each pixel (x, y) in the destination image, it then computes the coordinates of the originating pixel in the source image:

$$J_k(x, y) = I_k(f_x(x, y), f_y(x, y)) \tag{3.4}$$

Since coordinates $f_x(x, y)$ and $f_y(x, y)$ are typically fractional values, one can not simply copy the pixel value. Instead, values of neighboring pixels are fused by bilinear interpolation.

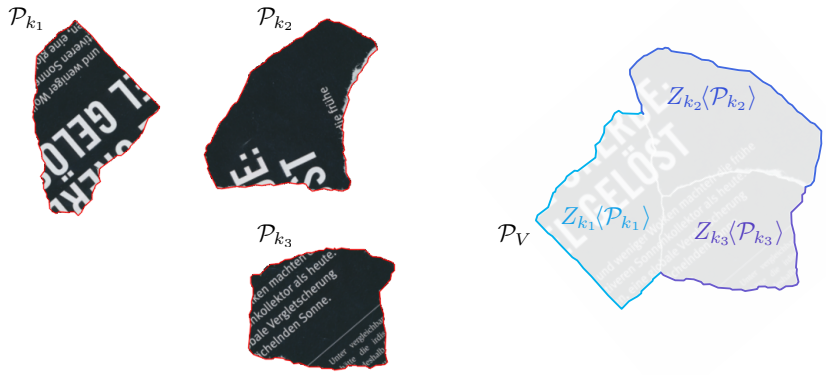


Figure 3.4: Example for a new artificial piece formed as representation of a group of three aligned pieces. **Left:** Individual pieces in their initial positions with polygons represented by $\hat{S}_k$ highlighted in red. **Right:** New artificial piece $\mathcal{P}_V$ for which each piece $\mathcal{P}_k$ is repositioned according to transformation Z_k. The piece's foreground is set transparent for better visualization.

3.7 Representing Groups of Aligned Pieces

Another key concept used in this thesis is the representation of a group of aligned pieces by a single *artificial piece*. For illustration, let $V = \{k_1, k_2, k_3\}$ be an index set representing three pieces $\mathcal{P}_{k_1}$, $\mathcal{P}_{k_2}$ and $\mathcal{P}_{k_3}$ in their initial position. Let Z_{k_1}, Z_{k_2} and Z_{k_3} be the corresponding transformations that determine the position of each piece in a tentative reconstructed document. An example for such a partial solution is given on the righthand side of figure 3.4. As illustrated in the figure, we represent this group of pieces as a single new piece $\mathcal{P}_V$. To this end we write

$$\mathcal{P}_V = (S_V, \hat{S}_V, I_V) \longleftarrow \texttt{group}(\{Z_k\langle\mathcal{P}_k\rangle \mid k \in V\}) \tag{3.5}$$

to indicate that a representation $\mathcal{P}_V = (S_V, \hat{S}_V, I_V)$ can be obtained via an operation `group`. We want to limit the discussion to an informal explanation of how this function can be implemented: Similarly as for individual pieces, the observed joint outer contour S_V can be obtained by the algorithm of Suzuki et al. [52]. For the approximated outer contour we reuse the support points of individual pieces. As visualized in the figure by different blue tones, each piece contributes a subset of its support points to define the closed polygonal curve $\hat{S}_V$. Finally, the foreground images of individual pieces are also combined into a single new image I_V. In the figure we set the piece's foreground transparent for better visualization.

Note that the method sketched here for the special case of $n = 3$ pieces generalizes to an arbitrary number of pieces.

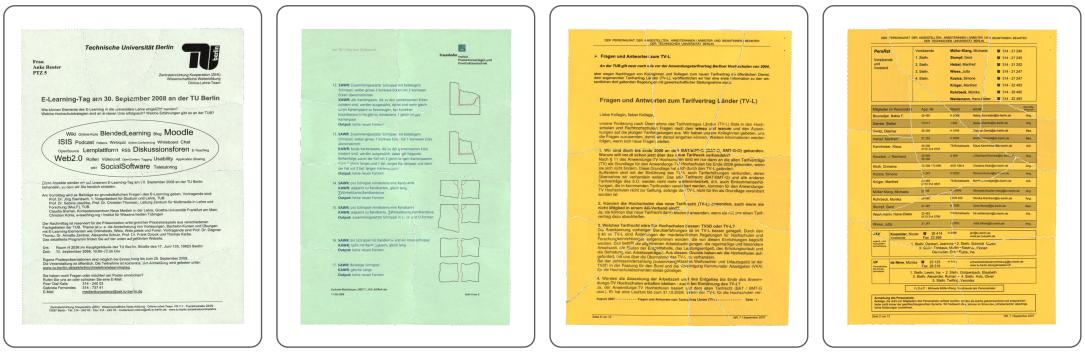


Figure 3.5: Examples for manually reconstructed pages from the *stieber500p* dataset.

3.8 Ground Truth from Manually Reassembled Pages

In order to be able to quantitatively evaluate the quality of reconstruction results, it is necessary to define a ground truth about the document pages first. For this purpose, we have created an annotation tool which allows human experts to reassemble individual pages digitally.

After placing all pieces on a canvas, the user is able to translate and rotate them – with adjustable accuracy – until all pieces fit together into a coherent whole. Some examples for manually reconstructed pages from the *bdw082010* and the *booklet* dataset are shown in figure 3.2 and figure 3.3, respectively. Since the authors of the *stieber500p* dataset could not provide us with an annotation, we scaled down their digitized pieces to fit into our annotation tool. As for our own two datasets we then proceeded by manually reassembling all 60 pages digitally. Examples for some manually reconstructed pages are shown in figure 3.5.

Positioning of Pieces in the Ground Truth

During this manual reconstruction process, we have kept track of each rigid transformation being applied to a given piece. Thereby we obtained a sequence of translations and rotations which are finally subsumed into a single transformation matrix. The resulting matrices G_k map pieces $\mathcal{P}_k$ to their final position in the coordinate system of the manually reconstructed document. The ground truth version of that piece is thus denoted by $G_k\langle\mathcal{P}_k\rangle$. For each piece $\mathcal{P}_k$ we store this accumulated rigid transformation G_k obtained from the user's repositioning.

Identifying Adjacent Piece-Pairs

Assuming a given page consists of N pieces, there are $\frac{N^2-N}{2}$ unique piece-pairs. Two pieces $\mathcal{P}_k$ and $\mathcal{P}_l$ are considered *adjacent* if there exists at least one *inlier*. An inlier is simply a pair of support points – one from each piece. For two points to be taken into consideration for a potential inlier we require that their Euclidean distance in the manually reconstructed document page is less than 5 pixels. That is, the *set of inliers* is a subset of all adjacent point-pairs:

$$S_{kl} \subseteq \big\{(i,j) \in \{1,\ldots,n(k)\} \times \{1,\ldots,n(l)\} \,\big|\, ||G_l \hat{\boldsymbol{s}}_j^l - G_k \hat{\boldsymbol{s}}_i^k|| \leq 5\big\} \tag{3.6}$$

Furthermore, we require that none of the support points is used for more than one inlier. To accomplish this, we determine one-to-one correspondences between pieces such that each support point from one piece has at most one counterpart on the other piece. The actual computation of S_{kl} is carried out in two steps: For each point i on piece $\mathcal{P}_k$ we first determine its closest point j on $\mathcal{P}_l$ with a Euclidean distance of less than 5 pixels. Among all points from the first piece choosing the same point j on the other piece we pick the one closest to j to form one inlier. Any point-pair that is not an inlier is in the following regarded as an *outlier*.

Our ground truth stores all piece-pairs $(\mathcal{P}_k, \mathcal{P}_l)$, $k < l$, whose set of inliers is non-empty. Two pieces are hereafter called *strongly adjacent* if they have at least four inliers.

Computing Expected Positions

For a given inlier indexed by (i,j) we also compute the offset vector

$$\boldsymbol{d}_{ij}^{kl} = G_k^{-1}(G_l \hat{\boldsymbol{s}}_j^l - G_k \hat{\boldsymbol{s}}_i^k) \tag{3.7}$$

$$= G_k^{-1} G_l \hat{\boldsymbol{s}}_j^l - \hat{\boldsymbol{s}}_i^k, \tag{3.8}$$

which specifies the *observed offset* of the j-th support point on piece $\mathcal{P}_l$, relative to the i-th point on $\mathcal{P}_k$, in the initial coordinate system of $\mathcal{P}_k$. These offsets are stored in our ground truth in addition to the list of all inliers.

Deviations from Observed Offsets

Based on the observed offsets of inliers it is possible to quantify the correctness of two pieces' *relative spatial configuration*. To illustrate the idea, consider two adjacent pieces $\mathcal{P}_k$ and $\mathcal{P}_l$. Let us assume that, by applying transformation Z_k to $\mathcal{P}_k$ and Z_l to $\mathcal{P}_l$, we correctly align the two pieces.

Using the observed offsets of one support point given the other, we compute the

residual vector $\boldsymbol{\epsilon}_{ij}^{kl}$ by:

$$\boldsymbol{\epsilon}_{ij}^{kl} = Z_k(\hat{\boldsymbol{s}}_i^k + \boldsymbol{d}_{ij}^{kl}) - Z_l\hat{\boldsymbol{s}}_j^l \tag{3.9}$$

$$= Z_k(G_k^{-1}G_l\hat{\boldsymbol{s}}_j^l) - Z_l\hat{\boldsymbol{s}}_j^l \tag{3.10}$$

If we choose $Z_k = G_k$ and $Z_l = G_l$ as transformations for the two pieces, the deviation from the ground truth is $\boldsymbol{\epsilon}_{ij}^{kl} = \mathbf{0}$. Intuitively, a residual vector of all zeros means that the pieces are aligned perfectly. Note that in this scenario, all inliers would entail the same residual vector. Obviously, if both transformations Z_k and Z_l deviate from these transformations only by the same rotation and translation, this does not change the pieces' relative positions, hence $\boldsymbol{\epsilon}_{ij}^{kl} = \mathbf{0}$ remains unaltered.

In practice, however, we are likely to encounter either partly misaligned or even spatially entirely disconnected pieces. Later in chapter 6 we will discuss how to quantify this degree of misalignment.

Chapter 4

Fundamental Techniques

4.1 Motivation

In this chapter we introduce fundamental techniques for the matching and identification of potentially adjacent document pieces. At its core, our reconstruction approach relies on robust features and supervised learning to disambiguate those very few correct piece-pairs from the disproportionately larger set of incorrect pairs. This chapter describes the evaluation of different geometric and content-based features.

Image features are also especially important for the automated solving of jigsaw puzzles. While most of the traditional variants dealt with the identification and matching of shapes [18, 60], most recent work focuses completely on robustly matching image content [2, 11, 23]. In recent years, the prevalent scenario was to focus on images that have been digitally cut into many square pieces. Due to the lack of shape information, more emphasis has been put on the development of image features. Obviously, as the number of puzzle pieces grows, so does the need for a more robust matching strategy in order to solve puzzles successfully.

In comparison to the matching of hand-torn document pieces, synthetically generated jigsaw puzzles oversimplify piece matching by strong assumptions: Correspondences of pixels between adjacent pieces are assumed to be known, and, due to the puzzle's virtual nature, the pieces' boundaries are not contaminated by noise. Under these premises, even simple similarity metrics such as the sum-of-squared color differences between neighboring boundary pixels yield satisfactory results. In our real-world setting, however, these assumptions typically do not hold. After digitization, document pieces show artifacts along their boundaries. Besides, due to the arbitrary shape of pieces, it may not even be possible to establish exact pixel-correspondences

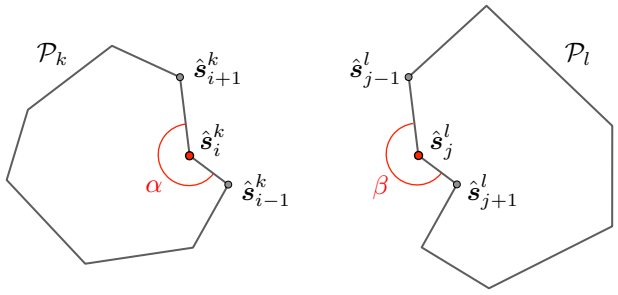


Figure 4.1: Schematic illustration of two pieces $\mathcal{P}_k$ and $\mathcal{P}_l$. In this example, the i-th support point on the left and the j-th one on the right form an inlier. Since these two points were adjacent in the original document, they should have a similar representation in terms of geometric and content-based features.

across their outer boundaries. These issues can lead to a sharp decline in the robustness of the matching.

This chapter is based on previous work [41] and is organized as follows: In section 4.2 we introduce a concept to evaluate whether two pieces match over a partial boundary. To accomplish this, we use a multimodal feature representation of support points. In sections 4.3 and 4.4 we then discuss the geometric and content-based features used for this purpose. Here we also introduce our new micro color feature based on the Fisher Vector (FV) framework. Besides being computationally very efficient we found this color encoding scheme to be extremely robust in the presence of noise. In section 4.5 we evaluate the effectiveness of the features on both, synthetically generated as well as real-world data examples, before concluding the chapter in section 4.6.

4.2 Identifying Matching Support Points

Since document pieces are irregularly shaped, we generally cannot presume to know their correct adjacent boundary regions in advance. Instead, our matching procedure for piece-pairs relies on a comparison of pairs of support points. To enable a principled way of comparing two pieces, we adopt a similar strategy as proposed in [8] for the digital reconstruction of hand-torn photos. The key idea is to compare pairs of support points (i.e., one point from each piece) in terms of a multimodal feature representation, which subsumes the pieces' contour-, shape-, and content-based characteristics within the vicinity of these points.

For an illustrating example, suppose that (i, j) corresponds to an inlier as shown in figure 4.1. Clearly, the polygonal approximation of both pieces is almost identical in

the immediate neighborhood around these two points, which is one indicator for them being locally compatible. From a geometrical perspective, the compatibility of pieces can be assessed in terms of the line segments attached to these points as they provide a simple way of characterizing the pieces' shape. Furthermore, given that both points stem from approximately the same location in the original document, it is reasonable to assume that the image content around these points is compatible as well.

In the following two sections we describe a set of eight features that we will make use of throughout the remainder of this work. By $\mathbf{\Phi}_{k,i} = \big[\mathbf{\Phi}_{k,i}[n]\big]_{n=1,\dots,8}$ we denote the descriptor of the i-th support point on the k-th piece. $\mathbf{\Phi}_{k,i}[n]$ is a fixed-dimensional representation of the support point regarding feature n. We describe the overall dissimilarity of two support points i and j from pieces $\mathcal{P}_k$ and $\mathcal{P}_l$ as the concatenation of dissimilarity values:

$$\mathbf{\Psi}_{k,l}(i,j) = \Big[K_n\big(\mathbf{\Phi}_{k,i}[n], \mathbf{\Phi}_{j,l}[n]\big)\Big]_{n=1,\dots,8} \tag{4.1}$$

Here $K_n : \mathbb{R}^{d_n} \times \mathbb{R}^{d_n} \to \mathbb{R}_0^+$ is a family of suitably chosen functions (one for each feature $n = 1, \dots, 8$) that computes non-negative dissimilarity values from two d_n-dimensional descriptors. In the subsequent sections 4.3 and 4.4 we explain how these dissimilarities are computed.

4.3 Contour- and Shape-Based Local Features

We start with a discussion of our first group of contour- and shape-based features. The first three features, namely the *angle feature* and the two *line features*, were introduced in [8]. These features merely rely on the points' immediate successors on the contour. Our *shape feature* completes this first group as the fourth feature.

4.3.1 Contour Description

To represent a given support point i, our first feature uses the enclosed angle on the inside of the polygon. As can be seen from figure 4.1, we characterize the contour's local curvature by α, which is the angle enclosed by the two line segments adjacent to that support point. Except for contour perturbations inevitably arising from noise, we expect a suiting complementary angle between two pieces whenever two support points form an inlier. Thus, to determine the dissimilarity of two points, we compute the absolute difference between angle α of the first point and the conjugate angle β of the second point.

The next two dissimilarity values consider the two line segments attached to each support point. For instance, as can be seen from the figure, the length of the first line segment from i to $i-1$ on piece $\mathcal{P}_k$ should be approximately equal to that from j to $j+1$ on $\mathcal{P}_l$. Assuming that $(i-1, j+1)$ is also an inlier, one would intuitively expect that the difference between their length amounts to 0. This gives us two features, one for each attached line.

4.3.2 Shape Feature

In order to complement the line and the angle features, we propose a contextual descriptor that encodes the piece's binarized foreground mask within a predefined spatial extent. Motivated by the shape context feature of Belongie et al. [4] we create a log-polar-like coordinate system centered around the support point. To capture the shape of the piece in the vicinity around that point, we define a radial grid that divides its neighborhood into angular and radial bins. We determine the weight of each bin as the percentage of foreground pixels intersecting with the respective bin. Only bins that almost exclusively cover the foreground region are assigned large weights, whereas sparsely populated bins obtain small weights.

Since a bin characterizes the ratio of foreground to background pixels in a certain area, one would expect that for inliers, the descriptors of corresponding support points accumulate to 1. This is because in the ideal case, inliers stem from pieces with complementary shapes, hence each foreground pixel on one piece implies a background pixel on the second piece. Conversely, when taking the inverse of exactly one of the two descriptors (e.g., binning foreground pixels for one piece and background pixels for the other), the resulting two descriptors should be mostly identical. In practice we compare shape descriptors using the Euclidean distance as dissimilarity measure.

Since the true upright direction of pieces is not known, we also need to ensure that our shape feature is invariant against rotation. To accomplish this, we normalize the descriptor by assigning radial bin indices relative to the line segment that is enclosed by the support point and its predecessing point. Note that the predecessor has to be chosen in opposite directions for two support points to be matched.

4.4 Content-based Local Features

Next we introduce a second group of features which capture the image foreground towards the inside of the polygon. By comparing two support points in terms of those

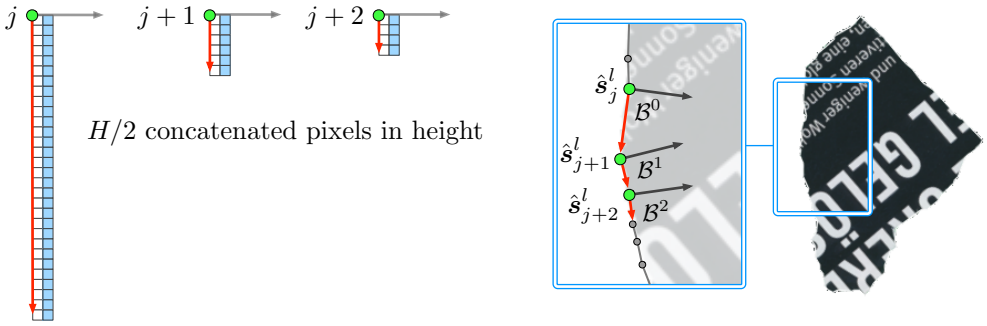


Figure 4.2: LCE is used to extract a rectified pixel layer ($L = 1$) as representation of point $\hat{\boldsymbol{s}}_j^l$ (stacking of light blue columns). If the line segment of the first point has insufficient length, additional support points are needed to obtain $H/2$ many rectified pixels. Source pixels stem from the local coordinate systems spanned by $\mathcal{B}^0$, $\mathcal{B}^1$ and $\mathcal{B}^2$. Embedded pixels in white color represent a void region ($V = 1$) and are omitted from the representation. Note that the remaining $H/2$ pixels in clockwise direction are not shown here.

features we strive to assess the local compatibility of pieces by their visual content. Although the image content may sometimes differ significantly on both sides of a tearing boundary, it is still reasonable to assume that inlier points share mostly compatible image contents within their immediate pixel neighborhoods, along the pieces' adjacent boundaries. Our aim is to determine whether the image content of one piece provides a consistent continuation of the other piece's content. For this purpose we use a set of color- and gradient-based local features.

Most of the widely used content-based (dis-)similarity measures that first come to mind require pixel-correspondences between the pieces' boundaries. For example, one measure often used for solving jigsaw puzzles is the sum-of-squared color differences between supposedly adjacent boundary pixels. We want to emphasize that establishing pixel-correspondences between two regularly shaped pieces is straightforward because they typically feature noise-free, straight boundaries of fixed length. In our real-world scenario, however, having to deal with arbitrarily shaped pieces turns out to complicate this task. To remedy this problem we next discuss a local coordinate embedding which straightens the image content of a piece along the inside of its polygonal curve. One appealing aspect of the proposed embedding is that it makes conventional compatibility measures directly applicable in our problem setting.

4.4.1 Local Coordinate Embedding

To obtain a rectified contour representation of fixed length, we introduce a method dubbed *local coordinate embedding* (LCE). The purpose of the LCE is the alignment of

pixels within the close proximity of a piece's boundary into a rectangular image patch. In analogy with synthetic pieces we further want the pixels to be laid out orthogonally to their boundary, i.e., its piecewise linear approximation. Consider the illustration in figure 4.2. To embed the piece's pixels in the vicinity of support point $\hat{\boldsymbol{s}}_j^l$, we always consider two subsequent points $\left\{(\hat{\boldsymbol{s}}_{j+m}^l, \hat{\boldsymbol{s}}_{j+m+1}^l)\right\}_{m\geq 0}$ at a time.

Denote by $\boldsymbol{v}_1^m = \hat{\boldsymbol{s}}_{j+m+1}^l - \hat{\boldsymbol{s}}_{j+m}^l$ the direction vector which is associated with the line segment enclosed by the m-th tuple. Based on $\boldsymbol{v}_1^m$ we then choose $\boldsymbol{v}_2^m$ as the one orthogonal vector being directed towards the inside of the piece. Intuitively, those two vectors define an orthogonal basis $\mathcal{B}^m$ and span a local coordinate system. We embed boundary pixels from that coordinate system into a straight pixel column (highlighted in light blue in the figure). By aligning $\boldsymbol{v}_1^m$ and $\boldsymbol{v}_2^m$ with the image axes we implicitly change the basis from $\mathcal{B}^m$ to an orthogonal basis of the local image coordinate system. Pixel intensities in the resulting straightened layer are computed from the source image using bilinear interpolation. Note that in the illustration in figure 4.2, a single layer is embedded ($L = 1$), which is offset by a one pixel void region ($V = 1$) from the boundary, in direction of $\boldsymbol{v}_2^m$.

Since line segments vary in length, so does the number of pixels that can be embedded along direction $\boldsymbol{v}_1^m$. In order to obtain a rectified patch of fixed height H, we thus adaptively adjust m (i.e., the number of local coordinate embeddings), until at least $H/2$ pixels have been stacked. Starting from $\hat{\boldsymbol{s}}_j^l$, the procedure is once applied for clockwise- and counterclockwise direction. The two resulting patches are each truncated to $H/2$ pixels and finally become stacked to define the rectified image patch of size $H \times L$. Notice that when traversing points in clockwise direction, the basis for LCE has to be adjusted accordingly, such that the y-axis is oriented upwards.

The representation obtained from LCE is a rectified image of $H \times L$ pixels that can be used for further processing. Therefore, it is possible to extract content-based feature descriptors exactly the same as if we were dealing with square jigsaw pieces. All of the following content-based features make use of the LCE representation.

4.4.2 Color Compatibility

A very simple – yet widely used – approach to determine the compatibility of two square image patches is based on the sum-of-squared color differences between their adjacent boundaries. Denote by P_k and P_l two patches of size $H \times H$. Without loss of generality, we assume that piece P_k is positioned to the left of piece P_l. To determine the pieces' compatibility we first extract two feature descriptors by stacking the color

values along the rightmost column of P_k and the leftmost column of P_l. We write:

$$\boldsymbol{\psi}(P; x, y) = \big[P(x, y, c)\big]_{c=1,\dots,3} \tag{4.2}$$

By c we refer to the 3 color channels of the CIELUV color space, x denotes a pixel column, and y is the pixel's vertical position within that column. According to the above definition, the stacking of color values along the pieces' adjacent boundaries can be formalized by $\boldsymbol{\psi}_L(P_l) = \big[\boldsymbol{\psi}(P_l; 1, y)\big]_{y=1,\dots,H}$, which comprises color values in the *leftmost* pixel column of the righthand piece[1]. Similarly, the *rightmost* column of the lefthand piece is represented by $\boldsymbol{\psi}_R(P_k) = \big[\boldsymbol{\psi}(P_k; H, y)\big]_{y=1,\dots,H}$. As in [11, 48] we determine the *color compatibility* of both pieces by computing the squared Euclidean distance between those two descriptors.

4.4.3 Mahalanobis Gradient Compatibility

The Mahalanobis gradient compatibility (MGC) has been proposed by Gallagher [23]. The author's key idea for the compatibility measure is to penalize changes in color gradients rather than penalizing changes in color directly. Here, the underlying assumption is that matching puzzle pieces should have continued edges across their adjacent boundaries. Besides, the author proposes to estimate the covariance between color channels to replace the Euclidean distance with the Mahalanobis distance.

4.4.4 Color Histogram

For our next descriptor we extract the color histograms over $\boldsymbol{\psi}_R(P_k)$ and $\boldsymbol{\psi}_L(P_l)$. As compatibility measure we tried different kernel functions and numbers of bins as shown later in section 4.5.

4.4.5 Fisher Vector Encoding

In the following we give a brief introduction to the Fisher Vector framework, as it forms the basis of our proposed micro feature. The Fisher Vector (FV) [30] provides a mechanism by which a variable number of features can be incorporated into a fixed-length signature, while retaining most of the discriminative power of the original features. Although the encoding incurs a loss of information – the original features cannot be recovered – it can be beneficial to encode features in this way. Normally, in order to successfully compare two feature sets directly, it is necessary to (i) establish

[1] We index both rows and columns from 1 to H.

feature correspondences, (ii) define a sensible similarity measure on those features, and (iii) incorporate the similarity of the individual feature pairs into a global measure of similarity.

The FV encoding provides a principled way to measure the similarity without steps (i) and (ii). It is based on the assumption that the distribution of features can be modeled through a generative process, i.e., it relies on a probability density function $p(\mathbf{x}; \boldsymbol{\lambda})$ that is parametrized by $\boldsymbol{\lambda}$. Typically, a Gaussian mixture model (GMM) is used for this purpose, in which case $\boldsymbol{\lambda} = \{\pi_1, \ldots, \pi_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K\}$ are the parameters of K mixture components. Here the π_k's and $\boldsymbol{\mu}_k$'s denote the components' priors and means, respectively. A commonly used simplification is to only consider diagonal covariance matrices Σ_k. By applying this standard assumption one can denote the diagonal of the matrix by $\boldsymbol{\sigma}_k^2$, which is the vector of variances.

In what follows, $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ represents our *feature set*, which has an associated likelihood function $u(\boldsymbol{\lambda}; \mathcal{X})$ with respect to the GMM model. The FV encodes the gradient $\nabla_{\boldsymbol{\lambda}'} \log u(\boldsymbol{\lambda}; \mathcal{X})$ regarding the model parameters of the log-likelihood of $\mathcal{X}$. Since the mixture priors have been shown [46] to add only little discriminative information, we exclude them from the gradient computation and only consider the gradients for $\boldsymbol{\lambda}' = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_K\}$. Note that the dimensionality of the FV does not depend on the number of features in $\mathcal{X}$. That is, given that the $\boldsymbol{x}_i$'s are d-dimensional vectors, their FV encoding has a fixed dimensionality of $2K \times d$.

After this brief summary it should have become clear that the FV does not require the explicit definition of feature correspondences and similarity measures on individual features. A histogram-based approach has the same property and indeed there is a connection between the FV and the histogram representation of a feature set. The FV can be regarded as a generalization of a bag-of-words (BoW) histogram [46]: If only gradients of the mixture weights π_k are considered for the FV computation, the FV becomes equivalent to a scaled and mean-centered BoW histogram with soft-assignment.

A benefit of employing the Fisher vector framework is that it eliminates the need of establishing pixel-correspondences while maintaining superior performance. To this end, let $\boldsymbol{\psi}_{1\times M}(P; x, y)$ denote a *micro patch descriptor* along a piece's boundary. Each such descriptor is the stacking of the color values on image patch P, at position (x, y), having a spatial extent of M pixels in height. An illustrative example for a piece $\mathcal{P}_k$ of size 10×10 pixels is depicted in figure 4.3. In the figure we show *micro patches* of size $M = 3$ pixels along the outermost pixel layer ($L = 1$).

For our experiments we perform a dense extraction of overlapping micro patches with 1 pixel stride. As a result, our feature set comprising all micro patches is given by

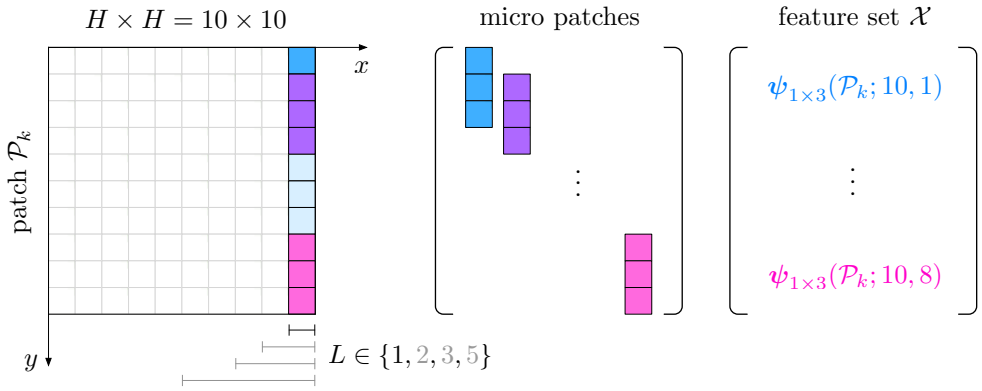


Figure 4.3: Illustration of micro patches extracted along the boundary region of a square jigsaw piece of size 10×10 pixels. This example shows micro patches of size $M = 3$ pixels, extracted with a stride of 1 pixel from the outermost layer. If multiple layers ($L > 1$) are used, each layer contributes the same number of descriptors to feature set $\mathcal{X}$.

$\mathcal{X}(P; x) = \{\psi_{1\times M}(P; x, y) \in \mathbb{R}^{3M} \,|\, y = 1, \ldots, H-M+1\}$. We then use the FV encoding to represent this set of local color descriptors as the concatenation of gradients relative to the log-likelihood of a Gaussian mixture model. Writing $\mathcal{X} = \mathcal{X}(P; x)$ in short we obtain the FV by

$$\psi_{\lambda'}(\mathcal{P}; x) = (\mathcal{G}^{\mathcal{X}}_{\boldsymbol{\mu}_1}, \ldots, \mathcal{G}^{\mathcal{X}}_{\boldsymbol{\mu}_K}, \mathcal{G}^{\mathcal{X}}_{\boldsymbol{\sigma}_1}, \ldots, \mathcal{G}^{\mathcal{X}}_{\boldsymbol{\sigma}_K}), \tag{4.3}$$

where $\mathcal{G}^{\mathcal{X}}_{\boldsymbol{\mu}_k}$ and $\mathcal{G}^{\mathcal{X}}_{\boldsymbol{\sigma}_k}$ are the d-dimensional gradient vectors with respect to a GMM with K mixture components (see [46] for details). We use the VLFeat library [58] for the computation of the *improved FV*, for which a power normalization is first applied in each dimension, before finally, the descriptor is ℓ_2-normalized.

By using the FV encoding we represent an entire pixel column of each piece through statistics of the deviation of its micro patch descriptors from a generative Gaussian mixture model. Similar as for the standard color compatibility we extract two descriptors $\psi_R(P_k) = \psi_{\lambda'}(P_k; H)$ and $\psi_L(P_l) = \psi_{\lambda'}(P_l; 1)$. Typically, the similarity measure of choice between two FVs is their dot product. However, since both vectors have unit length, there is no functional difference between their dot product and their Euclidean distance as both measures yield equivalent results up to an additive and multiplicative constant.

4.5 Evaluation

In this section we conduct experiments in order to optimize the parameter configuration of each feature. The choice of parameters for a given feature is considered to be

optimal if it yields the best possible classification performance, that is, if the configuration of the feature best distinguishes positive from negative examples.

We consider two evaluation scenarios:

- **Synthetic examples.** We first test the discriminativeness of the four content-based features on square jigsaw pieces. Therefore, we extract positive and negative examples from 10 randomly rotated versions of each page in *bdw082010* validation set, so as to avoid any bias for predominant horizontal or vertical image features. The positive examples are composed of two adjacent image patches, e.g., (P_k, P_l), which are sampled randomly from individual pages. Negative examples on the other hand are composed of two patches that are sampled from different pages.
- **Real-world examples.** In the second scenario we use annotated piece-pairs from the *bdw082010* dataset. Here we use inliers from adjacent piece-pairs as the basis for our positive examples. After applying the LCE to the inliers' support points, the resulting image patches are treated in the same way as boundary pixels from square jigsaw pieces. Negative examples are obtained in analogy to the synthetic case, using random piece-pairs from different pages. For each such pair we sample a pair of support points (outlier) which, after applying the LCE, provides us a pair of supposedly incompatible image patches.

4.5.1 Methodology

We compute the feature dissimilarity values, individually for each positive and negative example. Based on these dissimilarities, it is possible to jointly arrange all examples in one ranked list, which in turn allows one to plot a Receiver Operating Characteristic (ROC) curve. The area-under-curve (AUC) criterion is used as the measure for our comparison of different parameter configurations. Details on the ROC curve are discussed hereafter in section 4.5.3.

We distinguish two types of feature parameters: Those that we deem to be *sensitive* to real world data, and others that are likely to be *insensitive*. By labeling a parameter as "insensitive" we want to stress that we expect a fixed value to work equally well for synthetic and real-world problems. For instance, a Gaussian mixture model, which we employ as color model for the Fisher vector encoding, is most likely performing well on real shreds despite being trained on synthetic examples. Another example is the number of bins that has to be chosen for the color histogram. In accordance with the above reasoning we decided to adjust insensitive parameters on synthetic data.

In contrast to those examples just mentioned, our content-based features also involve other parameters that are most likely "sensitive" to characteristics that come into play on real-world shreds. For example, introducing an explicit "void region" in the LCE is beneficial when having to deal with real pieces. In order to optimize the features' robustness against noisy boundaries, we tune the width of the void region, which clearly should be an optimization driven by real-world examples.

4.5.2 Sensitivity & Specificity

In binary classification the aim is to map examples into a set of positive and negative instances. Commonly, the output of the classifier is a real value, and thus the classification boundary between the two classes must be chosen according to a threshold value. For example in the case of support vector machines (SVMs) [56], this threshold relates to the examples' distance from the hyperplane. Intuitively, this defines the separation boundary between the two classes in a high-dimensional vector space. With respect to this threshold, a classifier's prediction for any example can be either positive ($\hat{p}$) or negative ($\hat{n}$). To assess the correctness of predictions, we require the actual labels of instances, which are denoted by p (positive) and n (negative). For binary classification there are four outcomes to be considered: If the predicted label is $\hat{p}$ and the actual label is also p, the outcome is referred to as a *true positive* (TP). Conversely, a *false positive* (FP) has occurred if the actual label was n. If both the prediction and the ground truth agree on a given instance to be negative, the outcome is called a *true negative* (TN). Finally, positive examples that are incorrectly predicted to be negative are dubbed *false negatives* (FN).

For binary classification problems, the number of positive and negative examples are denoted by P and N, respectively. This leads to the definition of the *true positive rate* (TPR, or *sensitivity*), which is defined as the fraction $\frac{\mathrm{TP}}{\mathrm{P}}$. The *true negative rate* of a classifier (TNR, or *specificity*) on the other hand is $\frac{\mathrm{TN}}{\mathrm{N}}$.

4.5.3 Receiver Operating Characteristic (ROC) Curve

The *Receiver Operating Characterstic* (ROC) curve summarizes a classifier's performance for varying values of its discrimination threshold. To draw a ROC curve, one plots the false positive rate against the true positive rate, for any possible threshold. The false positive rate (1-specificity) is the fraction of misclassified negative examples. In practice, the threshold is adjusted to balance the true positive rate against the false positive rate depending on the requirements of the application. In spite of not using a classifier

here, one can still plot a ROC curve, individually for each feature modality. For this we use dissimilarity values as a confidence measure and determine the performance of each feature in terms of the AUC of its ROC curve.

4.5.4 Experiments

Next we conduct two groups of experiments. First we tune the insensitive parameters on our synthetic square jigsaw pieces for optimal performance. Afterwards, while holding the values of this first set of parameters fixed, we adjust all sensitive parameters on real pieces from {val} of the *bdw082010* dataset to evaluate the features' discriminativeness on real-world data.

Insensitive Parameters of Content-Based Features

The GMM for the Fisher vector encoding was trained using 10^6 micro patches that were randomly sampled from {train} of the *bdw082010* dataset. We used the VLFeat library [58] for training, and the covariance matrices were restricted to diagonal form.

Parameters for the color histogram (number of bins, B) and the FV (size of micro patches M, number of components in the GMM, K) were determined in separate experiments on {val}. To decide on the compatibility measure for the color histograms, we conducted experiments using a linear kernel, normalized- and unnormalized histogram intersection, and Chi-square [1]. The number of bins (per color channel) was chosen from $\{2, 4, 8, 16, 32, 64\}$. We found that $B = 32$, in conjunction with the unnormalized histogram intersection, worked the best and is hence used for all further experiments.

For the FV we evaluated micro patches of varying size, i.e., $M \in \{1, 2, 3, 5, 10\}$, in conjunction with different numbers of mixture components K for the GMM. We chose K to range from 1 to 16 in powers of 2. While most of the 25 experiments provided comparable AUC values, $M = 1$, $K = 4$ produced the best results. Notice that for this parameter configuration, micro patches correspond to individual pixels, hence the micro patch descriptors $\boldsymbol{\psi}_{1\times M}$ are only $d = 3$ dimensional (i.e., one value for each color channel). We want to emphasize that the FV, with $M = 1, K = 4$, thus yields a compact 24-dimensional color descriptor.

Sensitive Parameters of Content-Based Features

We organized experiments regarding the sensitive parameters in the following three groups, in which we compare the feature performances in different settings. First we

[1] For consistency with our other compatibility measures, we convert the resulting similarity into dissimilarity values.

discuss performances on square jigsaw pieces in order to provide a baseline for our experiments on our real-world examples.

(1) Varying Patch Height (H)

In our first set of experiments we evaluate the features' robustness depending on patch height H. As one would expect, longer pixel boundaries lead to higher discriminativeness, as is also reflected by the plots in the first row of figure 4.4 and 4.5. Despite its more compact representation, the FV performs on par with the MGC, and both features outperform the color compatibility and the color histogram feature. Since the number of available pixels is often limited (e.g., due to an increasing number of puzzle pieces), we assume a fixed height of $H = 40$ for the remainder experiments.

(2) Multiple Layers (L)

Next we make use of a multilayer pixel band closest to the piece's outer boundary. As dictated by L, we now consider up to 5 layers for pixelwise comparison, as well as for feature extraction (for color histogram, MGC, and FV), respectively.[1] As can be seen in the figures, this change has a detrimental effect on all features but the color histogram. However, we also note that the performance degradation for the FV is not as pronounced as for the pixelwise comparison and the MGC.

(3) Void Region (V)

With the plots in the last row we strive to evaluate the "robustness" of each individual layer. Therefore, we plot ROC curves for single layers that are offset from the outer boundary by V pixels. As can be seen in this plot series, the performances of features generally decline for pixel layers that are more distant in the source image. One can conclude from the plots, however, that the FV is the most reliable feature when the outermost pixel layers are not readily available ($V > 0$).

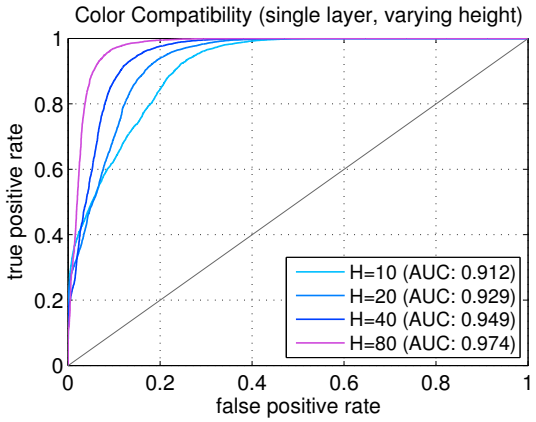
Summary

We want to emphasize that our 24-dimensional FV offers a more memory efficient representation than the other features: For $H = 40$, the color compatibility and the MGC respectively store 120 color values and color gradients. Note that even a sparse color histogram requires to encode at most 40 non-empty bins.

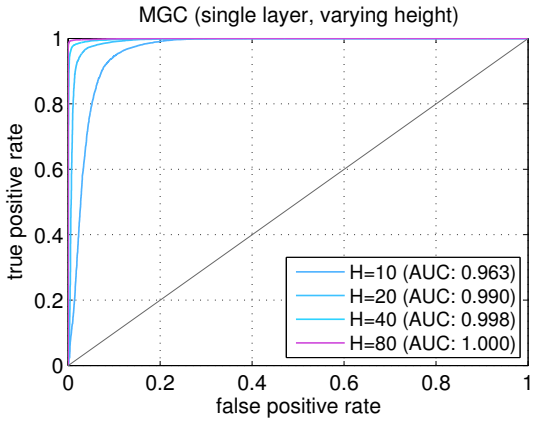
In summary, for real-world jigsaw puzzles one would anticipate the FV to outperform the other three features because in this scenario "void regions" become in-

[1] For the MGC we use gradients of all L layers for the computation of the covariance matrices.

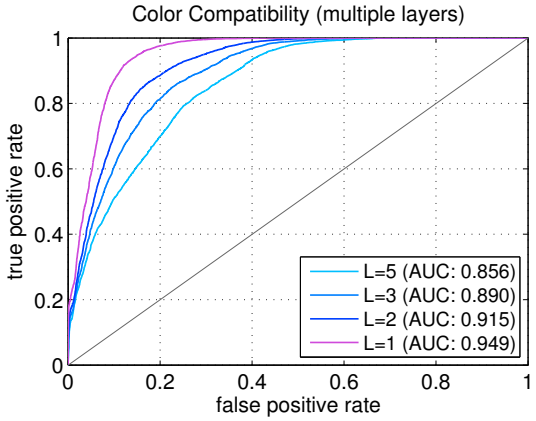
evitable. Considering the tearing process – be it by hand or a shredding machine – it becomes apparent that pieces are likely to suffer from material loss along their boundaries. Besides, the tearing not only proceeds along a paper's surface but also through the thickness of the document. Consequently, some of the outermost boundary pixels may be non-informative or even detrimental for matching and feature extraction.



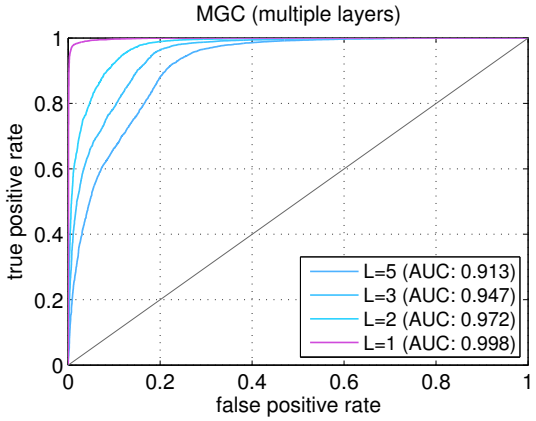
(a) Parameters: $L = 1, V = 0$



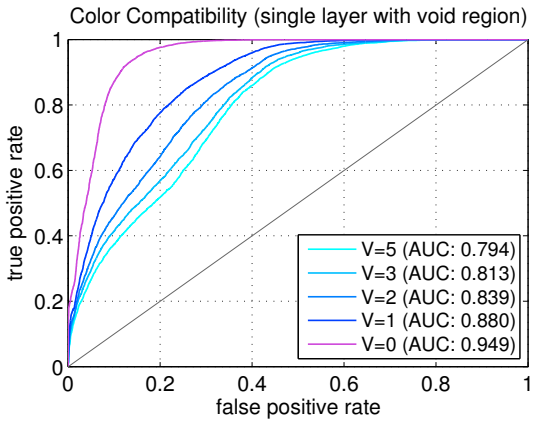
(b) Parameters: $L = 1, V = 0$



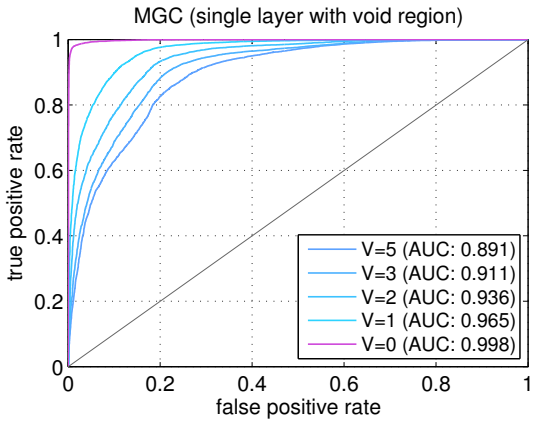
(c) Parameters: $H = 40, V = 0$



(d) Parameters: $H = 40, V = 0$

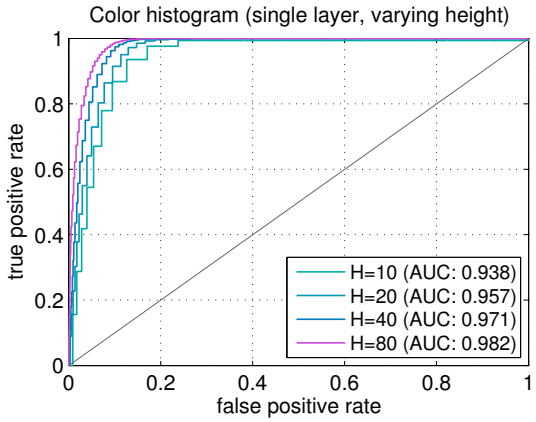


(e) Parameters: $H = 40, L = 1$

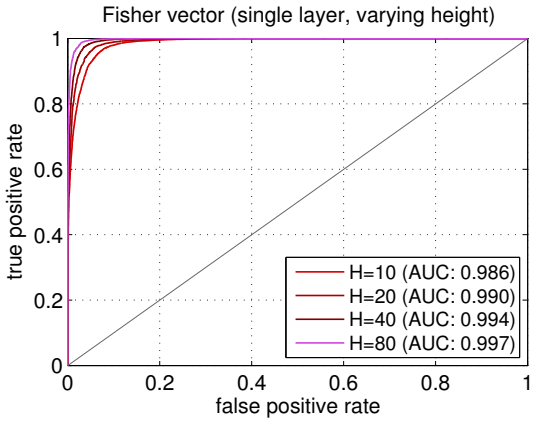


(f) Parameters: $H = 40, L = 1$

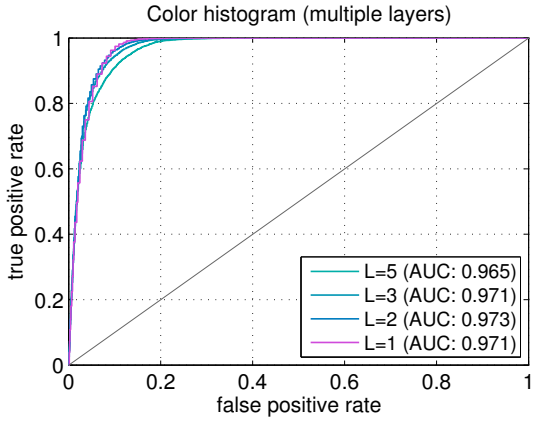
Figure 4.4: Feature comparison on square jigsaw pieces from the validation set. Three groups of experiments (rows) are conducted to evaluate the impact of the patch size, a multilayer representation, and how void regions influence performance. **Left:** *Color compatibility*. **Right:** *Mahalanobis gradient compatibility* (MGC).



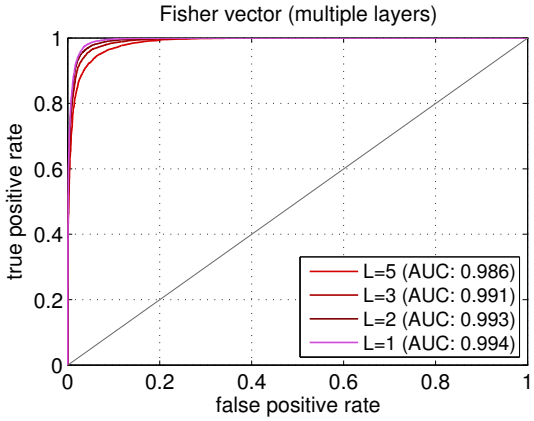
(a) Parameters: $L = 1, V = 0$



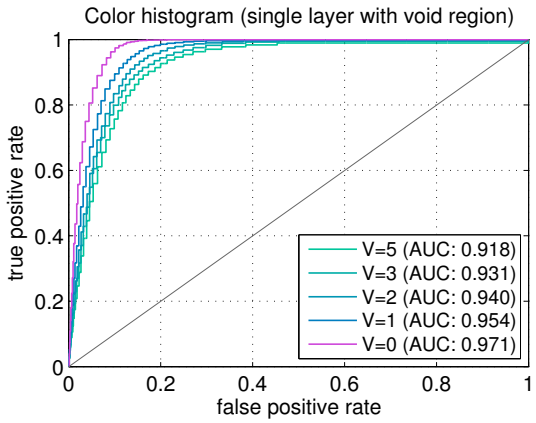
(b) Parameters: $L = 1, V = 0$



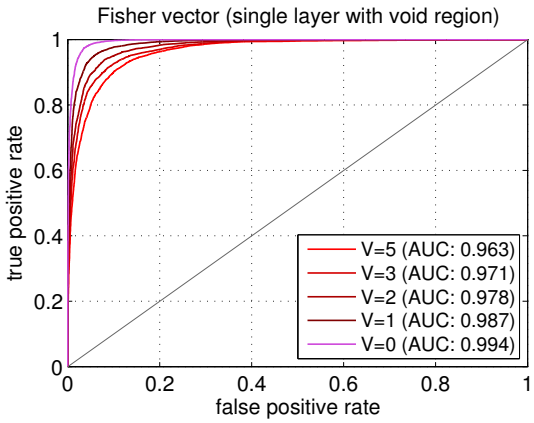
(c) Parameters: $H = 40, V = 0$



(d) Parameters: $H = 40, V = 0$



(e) Parameters: $H = 40, L = 1$



(f) Parameters: $H = 40, L = 1$

Figure 4.5: Feature comparison, continuing figure 4.4. **Left:** *Color histogram* (with $B = 32$ bins per channel). **Right:** *Fisher vector encoding on micro patches*, with height $M = 1$ and $K = 4$ Gaussian mixture components. The feature parameters for the FV (M, K) and the color histogram (B) were determined separately on the validation set (see text for details).

Parameters	AUC on the validation set of *bdw082010*			
(L,V)	Color compatibility	MGC [23]	Color histogram	Fisher vector (FV)
(1,0)	0.799	0.829	0.725	0.825
(1,1)	**0.812**	0.905	0.854	0.939
(1,2)	0.806	**0.920**	0.897	**0.959**
(2,2)	0.801	0.919	0.909	0.959
(3,2)	0.798	0.918	0.915	0.958
(5,2)	0.794	0.916	**0.920**	0.956

Table 4.1: Discriminativeness of content-based features for document pieces in terms of AUC depending on the number of layers (L) and void region size (V) in pixels. The best performance for each feature is printed in bold.

In our real-world experiments we use the same feature parameters that were previously determined on $\{\text{val}\}$ using synthetic jigsaw pieces. Also we re-use the Gaussian mixture model that was obtained from the synthetic training examples. In table 4.1 we summarize the results of our experiments. Two important observations can be made: First of all, despite having an "implicit void region" due to material loss, adding an explicit 2 pixel void region ($V = 2$) improves the classification performance (except for the standard color compatibility, where $V = 1$ works best). We attribute this to the fact that the outermost pixel layer is often contaminated by noise, e.g., by pixels that stem from the scan background. Second, we observe that the pixelwise comparison is outperformed by color histograms and the MGC, yet the latter requires only a single instead of five pixel layers for equivalent performance. Finally, we observe that the Fisher vector achieves almost a flat 4% increase in AUC. This is quite remarkable because its representation is more compact (by a factor of 5) compared to the two features based on pixel-correspondences.

Sensitive Parameters of the Shape Feature

From our four geometric features, the shape feature is the only one with intrinsic parameters regarding the log-polar-like grid. In practice, we found a wide range of parameters to work equally well. Thus, we empirically chose $r = 5$ pixels for the radius of the inner radial bin, with an expansion factor of $\gamma = 1.25$. That is, the radius of all bins $b \in \{1, \ldots, n_r\}$ is computed by $r \sum_{k=1}^{b} \gamma^{(k-1)}$, where n_r is the number of radial bins. We decided to use $n_r = 3$ to obtain approximately the same spatial extent as for our content-based features. For the number of angular bins we set $n_a = 15$ corresponding to a bin width of $24°$.

As for the content-based features we evaluate the discriminativeness of the shape

AUC on the validation set of *bdw082010*			
Angle	Line (front)	Line (back)	Shape feature
0.688	0.644	0.647	0.685

Table 4.2: Discriminativeness of geometric features for document pieces in terms of AUC.

feature and our three contour-based features. The results are reported in table 4.2. In direct comparison with the results from the content-based features, it becomes apparent that the geometric features are far less reliable. However, they do perform better than random guessing and thus can still be valuable when used in conjunction with each other. The next chapter investigates on exactly this aspect of how all of our features can be used jointly in a supervised learning approach.

4.6 Summary

In this chapter we have drawn a comparison between the methodology of matching square jigsaw pieces and real-world pieces. We have proposed a novel local coordinate embedding (LCE), which greatly facilitates the extraction of content-based features for real-world pieces. This enabled us to compare four content-based features in both application scenarios. Besides, we introduced a new color descriptor based on the Fisher vector encoding, which gives superior performance as compared to three other widely used features. We also showed experimentally that our proposed color descriptor is extremely robust in the presence of noise. Due to its low memory requirements and robustness, we deem it to be particularly useful in practical applications.

Part II

Reconstruction based on Binary Classification

Chapter 5

Adaptive Boosting and Geometric Signatures

5.1 Motivation

In the last chapter we introduced a local coordinate embedding that enables the extraction of content-based features in similar manner as for square jigsaw puzzles. However, finding a suitable representation for pieces is only one of the aspects one has to take into account for the reconstruction of hand-torn documents. In our application, the real-world nature of the problem dictates that the orientation of pieces is generally not known in advance. Since irregularly shaped pieces have no well-defined sides, there are virtually infinitely many spatial configurations that need to be considered. Clearly, testing every possible rigid transformation for the alignment of two pieces is infeasible because any pair of support points could be used (see section 3.5). Instead, we concentrate on identifying inliers [1]. This greatly reduces the number of transformations on the one hand, and on the other hand, it discards outliers, which otherwise would result in an incorrect spatial configuration if used for the alignment.

In this chapter we demonstrate a two-step approach for the identification of inliers: (i) We use binary classification to identify point-pairs having compatible feature representations. (ii) Based on geometric signatures we then perform a spatial verification. This postprocessing invalidates a large proportion of the misclassified point-pairs and thus further reduces the number of spatial configurations used throughout the reconstruction.

[1] Recall from section 3.8 that inliers correspond to pairs of support points across two pieces that are mutually close by in the manually reconstructed document.

In this chapter we discuss the adaptive boosting algorithm introduced by Freund and Schapire [20], which is more commonly called *AdaBoost*. Boosting is a supervised learning paradigm that creates highly accurate prediction rules by taking a majority vote from an ensemble of weak classifiers. Regarding our application, binary classification offers one way to decide whether contour points across two pieces are likely to stem from the same document location. We chose AdaBoost over other supervised learning approaches, e.g., support vector machines, for two reasons: First of all, despite that AdaBoost relies on only linear weak classifiers, the decision surface of the combined classifier is non-linear in the input space. Therefore, AdaBoost is capable of learning complex decision surfaces while maintaining the performance benefit of linear classifiers. Beside that, it provides an easy way of analyzing the reliability of individual features for the classification task, which is an integral component for the development and evaluation of potential new features.

The chapter is organized as follows: First we discuss fundamental aspects of adaptive boosting in section 5.2, before summarizing different classification settings in section 5.3. Afterwards, section 5.4 discusses how the model is trained. In section 5.5 we then explain how AdaBoost can be used to compute the importance of individual features for the classification task. The procedure described in section 5.6 performs a geometric verification of the classifier's predictions, which will be evaluated in section 5.7. Finally we conclude this chapter in section 5.8.

5.2 Fundamentals of Adaptive Boosting

In this section we give a brief introduction to the fundamentals of adaptive boosting, which for the largest part is a short summary of the comprehensive works of Freund and Schapire [20, 21]. Their works on this topic lay an excellent groundwork for a deeper understanding of adaptive boosting. The authors review many perspectives and analyses related to AdaBoost that also apply to machine learning in general.

The idea behind boosting is to combine multiple "weak" learning algorithms, which individually may perform only slightly better than random guessing, into a single "strong" classifier. The foundations of boosting lie in the Probably Approximately Correct (PAC) learning model due to Valiant [55], according to which such weak learners can be combined into an arbitrarily accurate classifier. Schapire [47] developed the first polynomial-time boosting algorithm in 1989. A year later, Freund [19] developed a more efficient algorithm, which, however, still suffered from several drawbacks and practical limitations. The algorithm reviewed here was later proposed in 1995, which

overcame many of the previous practical drawbacks.

Like other supervised learning methods for binary classification, AdaBoost takes as input a set of annotated training examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)$, where $\boldsymbol{x}_i$ is a feature descriptor in the *instance space* $\mathcal{X}$, and $y_i \in \mathcal{Y} = \{-1, +1\}$ denotes one of the two possible labels. AdaBoost is a greedy meta-algorithm which at training time repeatedly applies a weak learning algorithm for a fixed number of rounds $t = 1, \ldots, T$. One key concept of AdaBoost is that it is "adaptive" to the error of the weak learners trained in earlier rounds. That is, for each training example i, the algorithm keeps track of the example's weight during round t, which is denoted by $D_t(i)$. The weak learner's task then is to determine a *weak hypothesis* $h_t : \mathcal{X} \to \mathcal{Y}$ that best classifies training examples with respect to the weights in D_t, thereby putting more emphasis on misclassified examples. For the first round, the weights of examples are initialized uniformly, i.e., $D_1(i) = 1/m$. A weak hypothesis has a sample error ϵ_t:

$$\epsilon_t = \Pr_{i \sim D_t}[h_t(\boldsymbol{x}_i) \neq y_i] = \sum_{i : h_t(\boldsymbol{x}_i) \neq y_i} D_t(i) \tag{5.1}$$

In each round one chooses the hypothesis that minimizes the error for the current set of weights. Afterwards, AdaBoost assigns a weight factor α_t to the weak classifier

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right), \tag{5.2}$$

which intuitively measures its importance for the classification task. Next the set of all weights D_t is updated in order to (i) increase the weights for those examples that are misclassified by h_t, and (ii) decrease the weight of correctly classified instances. That is, the weights for the next round are set according to

$$D_{t+1}(i) = \frac{1}{Z_t} D_t(i) \exp\big(-\alpha_t y_i h_t(\boldsymbol{x}_i)\big), \tag{5.3}$$

where Z_t is a normalization factor chosen such that the weights in D_{t+1} sum to 1 in order to provide a distribution for the next training round.

Once all weak learners have been trained, the *strong classifier* for making predictions regarding a new instance $\boldsymbol{x}$ is obtained by:

$$H(\boldsymbol{x}) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right) \tag{5.4}$$

5.3 Matching Candidates

For the binary classification task at hand we consider pairs of support points from two pieces, i.e., one point from each piece. We call such a pair a *matching candidate*. With respect to the ground truth, these candidates can be grouped into two sets: The inliers (immediate neighbors from two adjacent pieces), which we regard as *positive examples*, and the outliers, which are used for training and testing as *negative examples*. Clearly, some negative examples are harder to distinguish from potential inliers than others. For this reason we decided to split our negative examples into three categories:

The first group of *cross-page* examples is composed of pairs of support points from two pieces that stem from different pages. This kind of negative example is the most representative of a document reconstruction task in which multiple pages are reassembled simultaneously because the large majority of point-pairs stems from cross-page piece-pairs. Naturally, the more pages are considered for reconstruction, the higher is the percentage of negative examples in this first group.

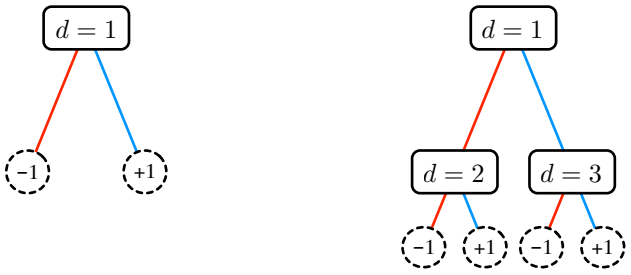
The second group of *intra-page* examples contains only outliers from non-adjacent pieces within a single page. Since an entire document corpus is likely to feature heterogeneous contents, one would assume that negative examples stemming from different pages are less compatible than those belonging to pieces from the same page.

Finally, our last group of *intra-page* examples comprises outliers among adjacent pieces, which are arguably the most challenging ones, since adjacent pieces are highly likely to feature similar image contents.

5.4 Weak Learners and Training Data

AdaBoost can be used with different types of weak learning algorithms. For example, *decision stumps* are a popular choice, which were used in the Viola-Jones face detection algorithm [59]. In this work the authors use AdaBoost not only because of its discriminative power, but more so to identify a reasonably small subset of critical features from an initially very large set of potential features.

As illustrated in figure 5.1a, a decision stump is a one-level decision tree that consists of only a single internal "decision node" as well as two leaf nodes. Since we use real-valued features in our work, a threshold is attached to the decision node, based on which examples are assigned to either the left or the right leaf node. Correspondingly, a prediction is made according to a single thresholding operation, which is based on only one input dimension. We use our multimodal dissimilarity vector $\mathbf{\Psi}_{k,l}(i,j)$ (see



(a) decision stump (depth 1)

(b) decision tree (depth 2)

Figure 5.1: Examples for two common weak learners. **Left:** Decision stump with a single decision node (root node) and two leaf nodes. **Right:** Two-level decision tree with a total of three decision nodes. Starting from the root node, examples are recursively split into the left branch (red) or the right branch (blue) until reaching a leaf node which assigns a class label ($+1$ or -1 for a positive or a negative prediction, respectively).

eq. (4.1)) to represent a given point-pair (i, j) across two pieces $\mathcal{P}_k$ and $\mathcal{P}_l$. Recall also from chapter 4 that each dimension of our 8-dimensional vector describes the points' dissimilarity in terms of one of our geometric and content-based features.

Decision stumps are not capable of utilizing multiple input dimensions in conjunction with each other, which limits the learner's discriminative power. For this reason we choose *decision trees* over these stumps as our weak learners. As illustrated in figure 5.1b, the difference between a two-level tree and a decision stump is that there are three instead of one decision nodes; leaf nodes still represent class labels, however, paths from the root to the leaves (branches) represent conjunctions of decisions based on individual features. The benefit of using trees over stumps as weak learners is that they can utilize features in conjunction with each other, hence they have potentially non-linear decision surfaces.

In all of our experiments we use trees with three decision nodes. Our set of positive training examples consists of all inliers from adjacent piece-pairs in {train} of the *bdw082010* dataset. The negative examples are obtained by randomly sampling $10{,}000$ point-pairs from non-adjacent intra-page piece-pairs.

5.5 Evaluation

In this section we first evaluate the importance of features for the classifier's decision, before conducting experiment regarding the performance of our classifier on the validation set. The importance of a feature can be regarded as an indicator for how well it is suited for discriminating inliers from outliers.

5.5.1 Feature Importance

To evaluate feature importances, we rely on the scheme implemented in the OpenCV library [1], which we want to sketch here briefly. The importance of a feature depends on its ability to split training examples correctly. Consider figure 5.1b. Starting from the root node, all examples are first split recursively until reaching the leaf nodes. For each of the three decision nodes, $d = 1, \ldots, 3$, we then proceed as follows: We decide on the feature $n_p(d) \in \{1, \ldots, 8\}$ and its corresponding threshold $T(n_p(d))$ that optimally splits the data. For classification tasks, common criteria used to determine the best split are the *entropy* and the *Gini impurity*. We compute the latter on the subset of examples assigned to a node's children, separately for each child node, and combine the resulting impurity values into a quality measure $q_d(n_p(d); D_t)$, to which we refer as the *quality of the split* (in node d). Note that the impurity value is computed based on "fractional examples" to account for the weight distribution D_t. The split with the highest quality among all possible choices of features for decision node d is called the *primary split*, which we represent by feature $n_p(d)$ and threshold $T(n_p(d))$.

Furthermore, for each decision node and each feature other than the one used for the primary split, we also compute a *surrogate split*. These splits resemble the primary split, however, their quality is upper bounded by $q_d(n_p(d); D_t)$ because the split results cannot be better than for the primary split (or else we would have chosen a different feature for the primary split). Just like for the primary split, each surrogate split is associated with a feature $n_s(d)$ and an optimally chosen threshold $T(n_s(d))$ for that input dimension. The quality of a surrogate split is expressed in terms of a scale factor. For each decision node d and for each choice of feature n we compute that quality scale factor as follows:

$$\gamma_d(n) = \frac{q_d(n; D_t)}{\max_{n'}\{q_d(n'; D_t)\}} \tag{5.5}$$

Obviously, $\gamma_d(n_p(d)) = 1$, and $\gamma_d(n_s(d)) \leq 1$. The equality holds only if the surrogate split for feature $n_s(d)$ splits the data exactly the same as the primary split.

The *relative decisive power* of a feature n within weak classifier h_t can then be computed by considering all decision nodes d and all possible splits (i.e., using primary and surrogate ones):

$$Q_t(n) = \frac{\sum_{d=1}^{3} \gamma_d(n) q_d(n; D_t)}{\sum_{n'=1}^{8} \sum_{d=1}^{3} \gamma_d(n') q_d(n'; D_t)} \tag{5.6}$$

Since $\sum_{n=1}^{8} Q_t(n) = 1$ holds, one can interpret $Q_t(n)$ as the quality of feature n

Geometric features	Angle	Line (front)	Line (back)	Shape feature
$I(n)$ w/ surrogate splits	0.1275	0.0924	0.0943	0.1183
Content-based features	Color compat.	MGC [23]	Color hist.	Fisher vector (FV)
$I(n)$ w/ surrogate splits	0.1427	0.1396	0.1147	0.1705

Table 5.1: Feature importance for the classification of pairs of support points using AdaBoost with two-level decision trees. To compute these values correctly, we used surrogate splits on all features. See text for details.

relative to the quality of all other features.

Finally, we accumulate the relative decisive power over all iterations and weight them according to the weak classifier's importance α_t. A weighting in terms of α_t is necessary as this tells us exactly how much weak learner h_t contributes to the overall strength of the final classifier. This leads to our definition of *feature importance*:

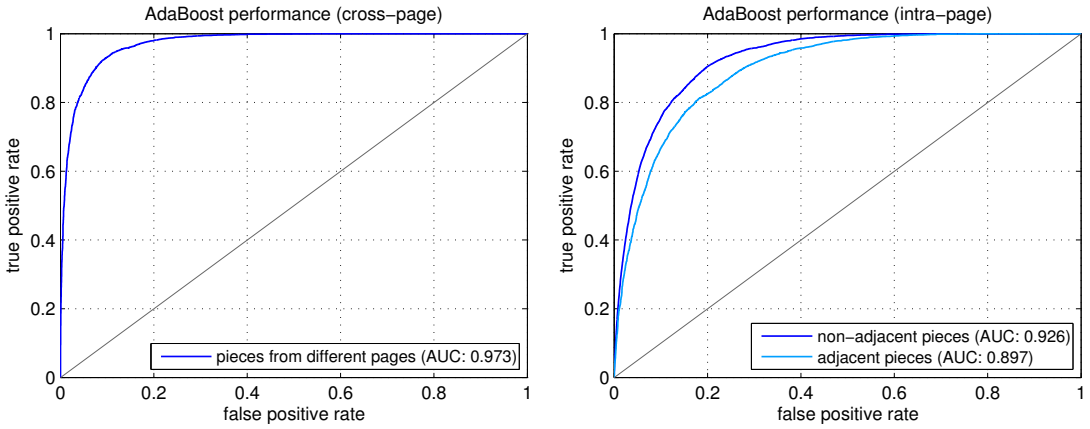
$$I(n) = \sum_{t=1}^{T} Q_t(n)\alpha_t / \sum_{t=1}^{T} \alpha_t \tag{5.7}$$

All feature importances are reported in table 5.1. We have obtained them from a model consisting of an ensemble of $T = 100$ weak learners trained with the OpenCV library [1]. We used a variant of AdaBoost called *GentleBoost* [22], which is less susceptible to outliers and thus often a preferable choice.

One can see from the table that the Fisher vector encoding, which achieved the best performance among all features in the experiments conducted in section 4.5, is also attributed the highest feature importance. Intuitively, AdaBoost greedily picks the most informative feature for the first weak learner, before "adapting" to the misclassified examples by updating the weights of all examples. It comes at no surprise that the Fisher vector encoding, which has shown the best individual performance of all features, is chosen at $t = 1$. Our geometric features, such as the points' inside angles and line lengths, showed mediocre discriminativeness in the experiments in chapter 4. Nevertheless, they still offer partly complementary information that can be used to obtain better classification results. Although their contribution to the classifier's performance is not as high, they still provide some utility for predicting examples correctly.

5.5.2 Classification Performance

Next we evaluate the classifier's performance for each of the three scenarios outlined before in section 5.3. Recall that in a ROC curve the false alarm rate is plotted against the recall as a function of the classifier's threshold. That is, using feature dissimilari-



(a) negative examples from different pages

(b) negative examples from the same page

Figure 5.2: Classification performance on cross-page negative piece-pairs (left) and intra-page piece-pairs (right). Using all features in conjunction with each other performs better than just using the FV by itself, which achieves an AUC of 0.959 (cross-page), 0.879 (non-adjacent intra-page), and 0.825 (adjacent pieces), respectively.

ties $\mathbf{\Psi}_{k,l}(i,j)$ to represent point-pairs obtained from $\{\text{val}\}$, we first create two disjoint subsets S_{pos} and S_{neg} that contain only positive and negative examples, respectively:

$$S_{pos} = \big\{\big(\mathbf{\Psi}_{k,l}(i,j), y\big) \,|\, y = +1\big\},\; S_{neg} = \big\{\big(\mathbf{\Psi}_{k,l}(i,j), y\big) \,|\, y = -1\big\} \tag{5.8}$$

Here, depending on the evaluation scenario, S_{neg} is either obtained through sampling outliers from (i) pieces of two different pages, (ii) non-adjacent pieces from the same page, or (iii) adjacent pieces from the same page. Given a classification threshold D, the set of true positives is obtained by:

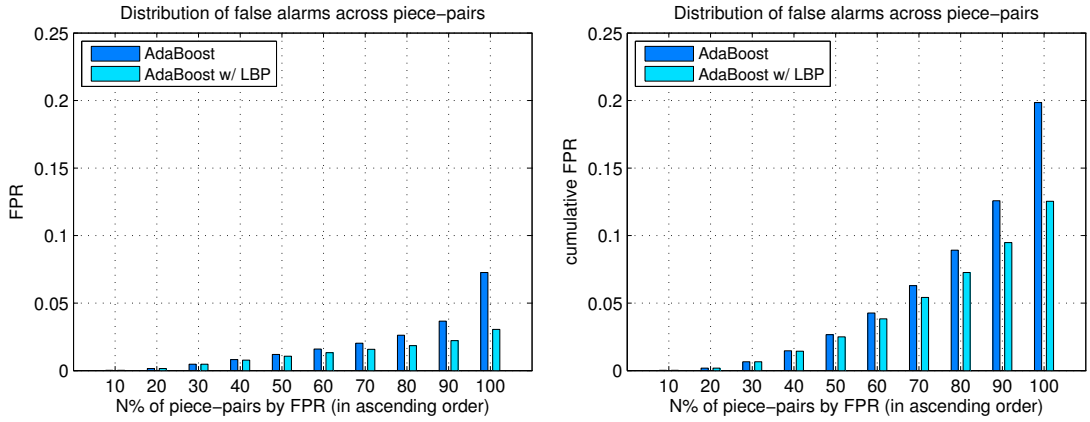
$$TP_D = \big\{\big(\mathbf{\Psi}_{k,l}(i,j), +1\big) \in S_{pos} \,|\, H(\mathbf{\Psi}_{k,l}(i,j)) > D\big\} \tag{5.9}$$

Similarly, all false positives are summarized in:

$$FP_D = \big\{\big(\mathbf{\Psi}_{k,l}(i,j), -1\big) \in S_{neg} \,|\, H(\mathbf{\Psi}_{k,l}(i,j)) > D\big\} \tag{5.10}$$

Each evaluation scenario uses its specific set S_{neg}, classifies the examples, and finally plots a ROC curve as shown in figure 5.2. As we have expected, the classifier performs best on the "easiest" subset containing only matching candidates from cross-page piece-pairs and steadily drops in terms of area-under-curve (AUC) for the more challenging data.

Since ROC curves report the true positive rate versus the false positive rate, it is



(a) false positive rate

(b) cumulative false positive rate

Figure 5.3: Enforcing LBP on the classifier results equalizes the distribution of false alarms across piece-pairs. **Left:** Non-adjacent piece-pairs from $\{\text{val}\}$ are grouped according to their false positive rate (FPR), in ascending order. **Right:** Cumulative FPR over groups, summed from left to right. See text for details.

possible to draw a fair comparison between experiments even if sample sizes are different. However, one obvious drawback is that ratios can be misleading when dealing with unbalanced problems for which one class contains substantially more examples than the other. For instance, in our scenario the number of negative examples heavily outweighs the number of positive examples. The validation set of the *bdw082010* dataset contains only 6396 inliers from 528 adjacent piece-pairs, as opposed to the approximately $6.8 \cdot 10^6$ outliers stemming from 1392 non-adjacent intra-page piece-pairs. Thus, in spite of choosing a considerably low false positive rate, we may still obtain a very large absolute number of false alarms. To quantify this effect, let D be the threshold associated with a recall of 0.5. Although the corresponding false alarm rate in this case is only 0.036, we would still have roughly $245{,}000$ false alarms across all non-adjacent pieces in $\{\text{val}\}$. There are two central problems caused by false alarms: (i) The computational complexity of the reconstruction procedure increases because more spatial configurations need to be considered. (ii) If an outlier is picked for the alignment of two pieces, this inevitably leads to an incorrect spatial configuration.

5.5.3 Locally Bounded Predictions

According to our empirical findings, the false positives are not distributed uniformly among all piece-pairs in the dataset. In fact, pieces with uniformly colored foreground along the tearing boundary cause most of these misclassifications.

To analyze this situation quantitatively, let $n(k)$ and $n(l)$ be the number of support points on the two pieces $\mathcal{P}_k$ and $\mathcal{P}_l$, respectively. The set of all positive predictions for this piece-pair is given by:

$$\hat{P}_{k,l}(D) = \{(i,j) \in \{1,\ldots,n(k)\} \times \{1,\ldots,n(l)\} \,|\, H(\mathbf{\Psi}_{k,l}(i,j)) > D\} \tag{5.11}$$

If not stated otherwise, this and all following experiments use the intra-page negative examples, i.e., outliers from non-adjacent pieces within the same page. The set of positive examples is always obtained from adjacent pieces. We set the threshold D to obtain a recall of 0.9, which corresponds to a false positive rate of $FPR_D = 0.198$ (see figure 5.2b). The false positive rate (FPR) regarding all examples in S_{neg} is then computed individually for each piece-pair, and the results are reported in figure 5.3. We group piece-pairs with regards to their FPR sorted in ascending order. As can be seen from the left bars in plot 5.3a, false alarms are distributed very unevenly across piece-pairs. If every pair would contribute equally to the false positive rate, the bars in the left plot would have equal height. The second plot in figure 5.3b is an alternative representation using the cumulative FPR.

To alleviate the effect of having very different classification performances across piece-pairs, our idea is to enforce an upper bound on the number of predictions. After classification, we keep only the top $\lfloor FPR_D(n(k)n(l)) \rfloor$ predictions from $\hat{P}_{k,l}(D)$ with the highest classification score, which retains less than 20% of all possible point-pairs. We call these postprocessed point-pairs the *locally bounded predictions* (LBP) to which we refer by $\hat{L}_{k,l}(D)$.

The effect of using $\hat{L}_{k,l}(D)$ (right bar) instead of $\hat{P}_{k,l}(D)$ (left bar) can be seen in figure 5.3. It becomes apparent that putting an upper bound on the number of predictions mostly equalizes the FPR across all piece-pairs. Another beneficial aspect is that it gives us a guarantee on how many point-pairs have to be considered. We can capitalize on this by performing a geometric verification, which is presented in the following section.

5.6 Geometric Signatures

The idea behind geometric signatures is to put each matching candidate into a spatial context with a second candidate. This enables the verification of the spatial configuration of points on one piece with regards to the spatial configuration of points on the other piece. Put differently, we discard a given candidate if there exists no other

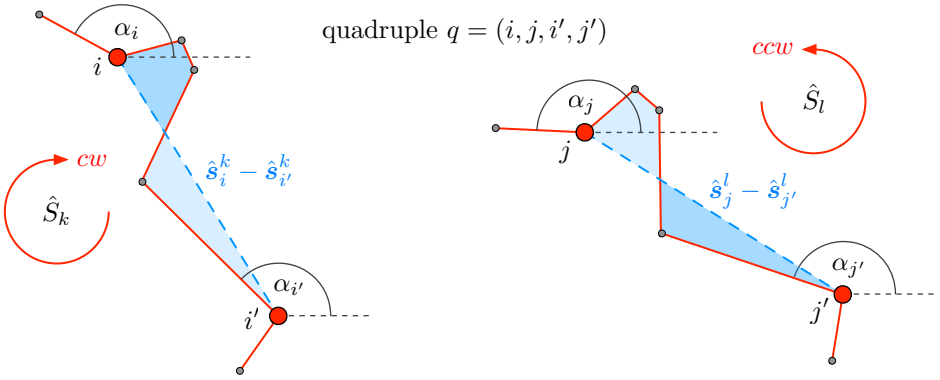


Figure 5.4: Schematic illustration of a quadruple $q = (i, j, i', j')$ comprising two inliers. **Left:** A small part of the contour of piece $\mathcal{P}_k$ is used to extract weak geometric constraints, with support points being traversed in clockwise (cw) direction. **Right:** A matching contour segment on piece $\mathcal{P}_l$ for which points are traversed in opposite direction (ccw).

candidate such that the two points on either piece have a similar spatial arrangement.

Consider the illustration in figure 5.4. It shows two inliers (i, j) and (i', j'), which together form a *quadruple* $q = (i, j, i', j')$. For each quadruple we create two *geometric signatures*, one separately for each piece. Signatures from the same quadruple are then compared with each other to verify whether their points are arranged consistently on both contours. For this purpose we introduce four *weak geometric constraints*:

As depicted in the figure, we characterize the relative positioning of two points by their Euclidean distance. The length of the line segment connecting two points i and i' can be seen as a weak constraint for what we expect to observe on the second piece, which we encode in the first signature component $\delta_1(i, i')$.

For an alternative characterization we determine the enclosed angle between the two line segments extending from each support point to its predecessor. As exemplified in the figure, we use the clockwise predecessor of i and i', on the contour of $\mathcal{P}_k$, to determine α_i and $\alpha_{i'}$, respectively. The second component of our signature $\delta_2(i, i')$ encodes the smallest sign-preserving enclosed angle between α_i and $\alpha_{i'}$.

Finally we characterize the local curvature of the polygon in between the points i and i'. As illustrated in the figure, we first create a polygonal chain from i to i' in clockwise direction, which is defined by a sequence of line segments connecting all pairs of consecutive points. We then form a closed polygon by appending the last line segment $\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_{i'}^k$ (dashed line). The key observation is that the polygon boundary can be self-intersecting. If this is the case, interior pixels of the polygon are dissected into two disjoint convex sets of image coordinates (e.g., the two blue areas in the figure).

These sets correspond to the foreground and background regions in the vicinity of the piece's outer boundary. We count the number of pixels on either side of the separating line $\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_{i'}^k$ and store the resulting values in $\delta_3(i, i')$ and $\delta_4(i, i')$. For signatures across the two pieces to match we require that foreground regions from one piece are complemented by background regions on the other piece and vice versa.

5.6.1 Compatibility of Signatures

We quantify the compatibility of two signatures by computing their element-wise differences. The resulting *residual value* for each weak geometric constraint $s \in \{1, \ldots, 4\}$ is denoted by $r_s = \delta_s(j, j') - \delta_s(i, i')$. Assuming that the two matching candidates correspond to noise-free inliers we would expect that their weak geometric constraints statistically cancel out each other[1]. That is, we assume a zero mean $\mu_s = 0$ for each of the four residual values. In practice, however, even inliers incorporate some noise, which we assume to follow a normal distribution. Accordingly, for each constraint we model $r_s \sim \mathcal{N}(\mu_s, \sigma_s^2)$. Two signatures are then called *consistent* if they satisfy

$$r_s \in [\mu_s - z_N \sigma_s, \mu_s + z_N \sigma_s] \tag{5.12}$$

for each weak constraint geometric constraint $s \in \{1, \ldots, 4\}$. In the above definition, z_N is a constant associated with the two-sided $N\%$ confidence levels of the normal distribution. For our experiments we empirically chose $z_N = 1.96$, which corresponds to a confidence interval that contains $N = 95\%$ of the data points. The mean and the standard deviation is computed separately for each constraint, based on the set of quadruples obtained from pairs of inliers within {train} of the *bdw082010* dataset. Note that this is equivalent to estimating a multivariate normal distribution, which is restricted to use a covariance matrix of diagonal form. In a preliminary experiment, we have computed all pairwise Pearson's correlation coefficients for the signatures' absolute residual values. Since the largest coefficient was 0.1363, neither of the constraints has a strong linear relationship with any other constraint. It is therefore unnecessary to estimate a full multivariate model because the signatures' residual values are almost uncorrelated.

[1] Note that the residual value for constraint $s = 2$ is computed as the smallest sign-preserving enclosed angle between $\delta_2(i, i')$ and $\delta_2(j, j')$.

Algorithm 5.1: Verification via geometric signatures (`sig-verify`)

Input : Classification threshold D
Quantifiers : Piece indices (k, l)
Parameters : Constant for confidence interval z_N, number of required votes n_P, normal distribution parameters (μ_s, σ_s), $s \in \{1, \ldots, 4\}$
Output : Set of inlier candidates A

Initialization

`/* set of verified matching candidates */`
$A \leftarrow \emptyset$
`/* set the number of votes to zero for each matching candidate */`
$V(i, j) \leftarrow 0$

Verification by voting

`/* compute positive predictions */`
$\hat{P}_{k,l}(D) = \{(i, j) \in \{1, \ldots, n(k)\} \times \{1, \ldots, n(l)\} \mid H(\mathbf{\Psi}_{k,l}(i, j)) > D\}$
`/* postprocess classification results w/ LBP */`
retain the top elements of $\hat{P}_{k,l}(D)$ *in* $\hat{L}_{k,l}(D)$ (see section 5.5.3)
`/* candidate voting */`
foreach *unique quadruple* $(i, j, i', j') \in \hat{L}_{k,l}(D) \times \hat{L}_{k,l}(D)$, $i \neq i'$, $j \neq j'$ **do**
  if $\delta_s(j, j') - \delta_s(i, i') \in [\mu_s - z_N\sigma_s, \mu_s + z_N\sigma_s]\ \forall s \in \{1, \ldots, 4\}$ **then**
    increment both $V(i, j)$ *and* $V(i', j')$ *by* 1
`/* pick candidates with sufficient number of votes */`
foreach $(i, j) \in \hat{L}_{k,l}(D)$ **do**
  if $V(i, j) \geq n_P$ **then**
    $A = A \cup \{(i, j)\}$

return A

5.6.2 Verification Procedure

The whole geometric verification procedure is summarized in algorithm `sig-verify`. Given a piece-pair and a classification threshold D as the input, the procedure determines the set of geometrically verified matching candidates that accumulate a sufficient number of "votes" from the other candidates. In the verification procedure we denote the number of votes for a matching candidate (i, j) by $V(i, j)$. The assumption is that two candidates verify each other only if both correspond to inliers. Although this assumption certainly does not always hold true, we conjecture that two outliers

are less likely to entail consistent signatures as compared to two inliers. Therefore, we expect inliers to accumulate a higher number of votes than the few remaining outliers, which are usually scattered almost randomly across the pieces' contours.

Beside the parameters of the normal distributions, the procedure also depends on the required number of votes n_P. The optimal value of n_P is evaluated experimentally in the next section.

A final note on the implementation: Since quadruples are meant for spatial verification, we empirically chose a minimum distance for point-combinations. We only create a quadruple from two matching candidates if on both pieces the length of the line segments is larger than 25 pixels, i.e., if $\delta_1(i, i') > 25$ and $\delta_1(j, j') > 25$ hold.

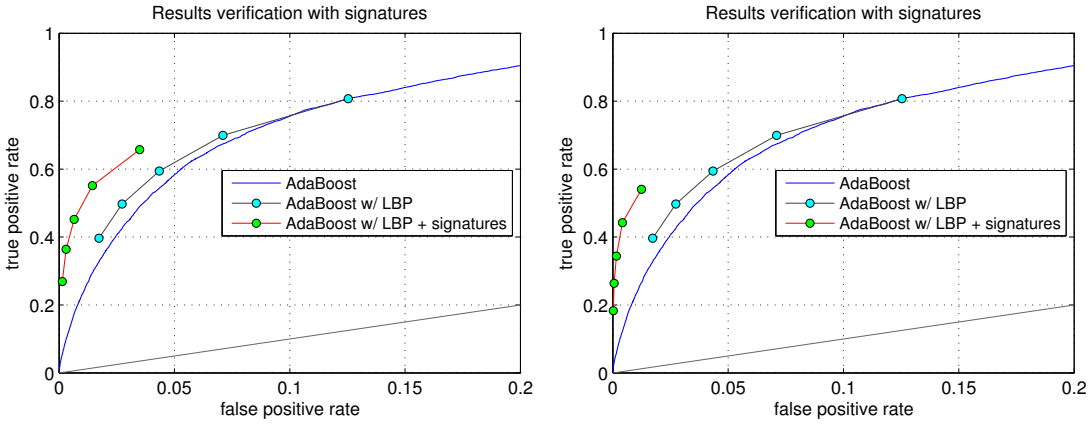
5.7 Evaluation

We now conduct experiments to investigate how locally bounded predictions and geometric signatures influence our classification results. We stick to the evaluation scenario also used for the plots in figure 5.2b. That is, we once again consider negative examples obtained from non-adjacent intra-page piece-pairs and draw a comparison to the results obtained from AdaBoost without further postprocessing.

5.7.1 Performance after Verification with Signatures

As the last part of our evaluation we assess the impact of using geometric signatures for the postprocessing of AdaBoost's classification results. For this purpose, we conduct two experiments which differ only in the number of "votes" each matching candidate needs in order to be considered as an *inlier candidate*. We choose $n_P = 1$ votes for our first experiment, for which the results are reported in figure 5.5a. The dots correspond to the equivalent true positive and false positive rates for recall levels of 0.9, 0.8, 0.7, 0.6 and 0.5, once after (i) having applied LBP and (ii) using LBP and geometric signatures in conjunction with each other. As can be seen from the plot, equalizing the FPR across piece-pairs (AdaBoost w/ LBP) improves the classification performance. However, the biggest performance gain is obtained by using geometric signatures in addition (AdaBoost w/ LBP + signatures). Since now only verified point-pairs can count towards the recall, the equivalent recall values are typically a bit lower as compared to using LBP by itself. However, the overall classification performance improves because the false positive rate is significantly lower for any given level of recall.

In analogy with this first experiment, the plot in figure 5.5b shows the TPR and FPR for matching candidates that require at least $n_P = 2$ votes in order to be retained.



(a) number of votes $n_P = 1$ **(b)** number of votes $n_P = 2$

Figure 5.5: Performance after postprocessing with locally bounded predictions and geometric signatures. The blue curve (AdaBoost) is taken from the plot in figure 5.2b. **Left:** Matching candidates require at least $n_P = 1$ vote in order to be retained. **Right:** $n_P = 2$ votes are required for verification.

As one would expect, the false alarm rate drops even further, however, at the cost of recall. We would like to note that the recall refers to the percentage of inliers that are identified successfully. A value less than 1 does not indicate that the document can not be reconstructed entirely. As we will see shortly in the next chapter, a single inlier is already sufficient to identify the correct spatial configuration of any given piece-pair. Nevertheless, higher recall values are always better because some piece-pairs have very few inliers. In practice we found a recall of at least 0.4 to be sufficient.

It should also be emphasized that putting an upper bound on the number predictions with LBP speeds up the verification procedure. This is because the number of geometric signatures grows quadratically in the number of the classifier's positive predictions. By limiting the number of predictions from each piece-pair we know that $|\hat{L}_{k,l}(D)| \leq \lfloor FPR_D(n(k)n(l)) \rfloor$ holds, which in turn limits the number of quadruples that need to be examined throughout the verification. For our choice of $FPR_D = 0.198$ (see section 5.5.3) LBP keeps at most 20% of all point-pairs.

In the following we always use classification results postprocessed with both LBP and geometric signatures (with $n_P = 1$). If not otherwise specified, we use a threshold D corresponding to a recall of 0.9 (before LBP + signatures). The set of all *inlier candidates* obtained from algorithm `sig-verify` forms the basis for recovering the correct spatial configuration of piece-pairs, which will be discussed in the next chapter.

5.8 Summary

In this chapter we have demonstrated how adaptive boosting can be used to identify a set of inlier candidates. Based on our multimodal feature representation introduced in the previous chapter, we trained a binary classifier from a set of annotated training examples. The performance of our classifier has then been analyzed in three different scenarios of varying difficulty. Moreover, we proposed a two-step postprocessing to discard incorrectly classified candidates: After enforcing an upper bound on the number of predictions per piece-pair, geometric signatures were used to perform a spatial verification. Our experiments show that we are capable of identifying more than 55% of all inliers at a false positive rate below 1.5%.

Chapter 6

Recovering Spatial Configurations of Piece-Pairs

6.1 Motivation

In this chapter we present an algorithm to recover the correct spatial configuration of piece-pairs. For each of the inlier candidates identified in the previous chapter we first compute an initial rigid transformation H that aligns one piece with the other. Most often this initial estimate is not very accurate and hence needs to be refined. For this purpose we use a coarse-to-fine *hill-climbing* optimization. Starting from the pieces' spatial configuration entailed by the initial estimate H we apply a local search. Our algorithm attempts to find a more accurate solution by adjusting the rotation angle of H in an iterative fashion. Since the translation remains unaltered throughout the optimization, our approach performs a search on a one-dimensional grid. In order to speed up the optimization we perform the local search on grids of increasingly finer scales.

Parts of the method discussed in this chapter have been described in [42, 43]. The outline of the chapter is as follows: First in section 6.2 we discuss how initial estimates for rigid transformations can be obtained from individual matching candidates. Our hill-climbing algorithm is then presented in section 6.3. Here we demonstrate how geometric and content-based compatibility scores can be incorporated into its objective function, based on which we attempt to find the optimal rotation angle. In section 6.4 we present our evaluation methodology, and we also explain how to assess the correctness of spatial configurations regarding our ground truth. Finally, section 6.5 concludes the chapter with a brief summary.

6.2 Initial Estimates for Spatial Configurations

Throughout this chapter, a rough estimate for the spatial configuration of two pieces $\mathcal{P}_k$ and $\mathcal{P}_l$ is going to be based on a single matching candidate (i, j). Recall from the discussion in section 3.5 that a single point-pair is sufficient to determine the translation of transformation H. Let us reconsider eq. (3.2) again:

$$H_{k,l,i,j}(\omega) = T(\hat{\boldsymbol{s}}_i^k)R(\omega)T(-\hat{\boldsymbol{s}}_i^k)T(\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_j^l) \tag{6.1}$$

Applying this transformation to piece $\mathcal{P}_l$ superimposes support point j with point i into a single point, which is used as the rotation center. Since H is parametrized only in ω, the spatial configuration of this piece-pair solely depends on the value of this parameter.

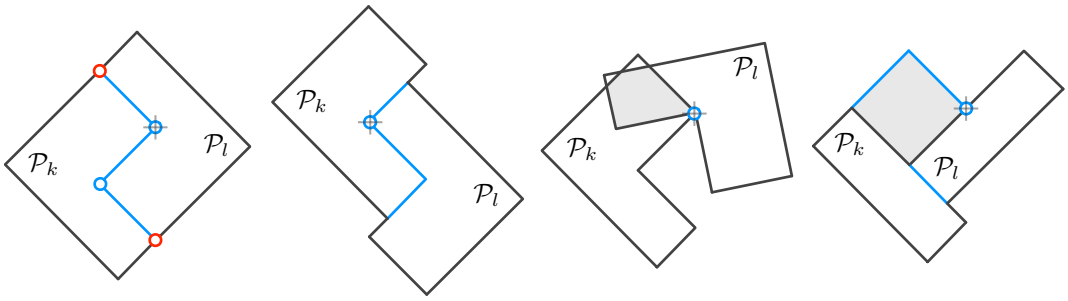
To obtain an initial estimate for ω, we use the immediate successors of support points i and j, in clockwise and counterclockwise direction on the contours of $\mathcal{P}_k$ and $\mathcal{P}_l$, respectively. We compute our estimate $\hat{\omega}$ as the enclosed angle between the line segment from i to i' and the one from j to j'. Although this initial estimate $\hat{\omega}$ is not necessarily an optimal solution, it is most often quite near to the optimum. Therefore, providing this estimate as the starting point to our algorithm limits the search range and also makes our method less susceptible to getting stuck in a local optimum.

In the following we represent the transformation matrix briefly by $H_e(\hat{\omega})$, where $e = (k, l, i, j)$ is the qualifier denoting the pieces and the points taken as basis for the calculation. We need to distinguish transformations in that way because (i) there are multiple inlier candidates that need to be considered and (ii) our reconstruction algorithm requires an unambiguous representation of piece-pairs.

6.3 Optimizing Spatial Configurations through Hill-Climbing

Hill-climbing is a mathematical optimization method that performs local search, starting from an arbitrary point in the solution space $\mathcal{X}$. It proceeds in iterative fashion by changing single elements of the current solution $\boldsymbol{x} \in \mathcal{X}$. This sets hill-climbing apart from gradient descent approaches, which generally adjust all solution parameters at the same time, in direction of the negative gradient of the objective function. Beside that, in contrast to gradient descent, hill-climbing can be applied even if the objective function is non-differentiable.

If the solution space is defined by a discrete grid, the search strategy in hill-climbing simplifies to examining neighboring grid points. The algorithm is widely applicable



(a) correct **(b)** incorrect **(c)** incorrect **(d)** incorrect

Figure 6.1: Illustration of different spatial configurations of two pieces. The cross marks the rotation center and the blue line represents the adjacent boundary segment between the two aligned pieces. Blue circles: inlier candidates obtained from AdaBoost /w LBP + signatures (see chapter 5). Red circles: invalidated matching candidates.

as it attempts to solve the general problem of maximizing an objective function $f(\boldsymbol{x})$ over either a continuous or a discrete solution space $\mathcal{X}$. Starting from an initial solution $\hat{\boldsymbol{x}} \in \mathcal{X}$, the algorithm iteratively improves the current solution until no further progress can be made.

In our problem setting, the only quantity to be optimized is rotation angle ω. That is, the objective function is of the form $f_e : \Omega \to \mathbb{R}$, where Ω is the domain of ω and $e = (k, l, i, j)$ is the quantifier later required to distinguish different initial estimates.

6.3.1 Measuring the Compatibility of Spatial Configurations

Our objective function for the hill-climbing optimization builds on compatibility scores that characterize a spatial configuration of pieces in four ways: (i) *length of adjacent contours*, (ii) *absolute intersection*, (iii) *relative intersection*, and (iv) *classification results*.

A few illustrating examples for different spatial configurations are depicted in figure 6.1. Figure 6.1a shows the optimal configuration of two pieces. In this example, pieces have a long adjacent boundary and do not overlap each other. Besides, there are two positive predictions from AdaBoost, which thereafter also passed the verification through LBP and geometric signatures (see chapter 5). We dubbed the set of *verified* matching candidates obtained from running procedure `sig-verify` the *inlier candidates*. The second example in 6.1b shows an incorrect spatial configuration with only one inlier candidate. In spite of being somewhat compatible in terms of the criteria (i)–(iii), the compatibility scores should be no higher than for the correct configuration in the first example. The other two examples in figures 6.1c–6.1d illustrate that the two overlap criteria are reliable indicators for incorrect spatial configurations.

For our first compatibility score we quantify the *length of adjacent contours* by:

$$ {}^{1}m_e(\omega) = \left|\texttt{polyline}(S_k) \cap \texttt{polyline}(H_e(\omega)S_l)\right| \tag{6.2} $$

Recall that one of the equivalent representations of S_l was a $3 \times n(l)$ matrix, which stores support points of $\mathcal{P}_l$ in homogeneous coordinates as column vectors. Let $H_e(\omega)$ be the 3×3 transformation matrix that aligns contour points in S_l with those in S_k. Thus, the columns of vector-matrix product $H_e(\omega)S_l$ are homogeneous point coordinates of $\mathcal{P}_l$'s contour points after aligning to $\mathcal{P}_k$. By `polyline` we refer to a procedure that determines the set of boundary pixels along the closed polygonal curve. In our implementation we first use the Bresenham algorithm to draw contour lines that are 2 pixels wide and then compute the intersection between the two resulting pixel masks.

Next we characterize the degree of intersection between two aligned pieces in two ways. For this purpose we introduce the procedure `fillpoly`, which determines the set of interior pixels bounded by a polygonal curve. For this compatibility score we compute the intersection of the two sets obtained from our aligned pieces:

$$ {}^{2}m_e(\omega) = \left|\texttt{fillpoly}(S_k) \cap \texttt{fillpoly}(H_e(\omega)S_l)\right| \tag{6.3} $$

$$ {}^{3}m_e(\omega) = {}^{2}m_e(\omega) / \left\lfloor |\texttt{fillpoly}(S_k)|, |\texttt{fillpoly}(H_e(\omega)S_l)| \right\rfloor \tag{6.4} $$

Besides the *absolute intersection* defined in eq. (6.3) we put that quantity in a different context by also computing the pieces' relative intersection. In eq. (6.4) we obtain the *relative intersection* by also taking into account the minimum number of foreground pixels from both pieces.

Our last compatibility score is based on the *classification results*. It counts the number of inlier candidates that are in close proximity to one another *after* the alignment. Formally, this can be written as:

$$ {}^{4}m_e(\omega) = \log\left(\left|\{(i,j) \in A_{1:1} \,|\, ||\hat{\boldsymbol{s}}_i^k - H_e(\omega)\hat{\boldsymbol{s}}_j^l|| < 5\}\right|\right) \tag{6.5} $$

First, we use algorithm `sig-verify` to determine set A. Recall that this procedure involved three steps: (i) classification of matching candidates with AdaBoost, (ii) identification of the top-ranked elements in the list through LBP, and (iii) verification of matching candidates with geometric signatures. Since those steps do not depend on the pieces' orientation, we can precompute the set A *before* the alignment is made. If ω changes, the only thing that has to be reevaluated is the adjacency relationship between point-pairs.

The second step is to apply the same method used for the inference of inliers for our ground truth (see section 3.8). That is, before computing ${}^4m_e(\omega)$, we determine a subset $A_{1:1} \subseteq A$ of one-to-one correspondences across the two pieces. This ensures that each support point contributes to at most one of the pairs (i, j) in eq. (6.5). Finally, note that the logarithm is used to limit the impact of unicolor contour regions, which tend to produce many unreliable inlier candidates.

6.3.2 Objective Function

Next we define our *objective function*. Since our goal is to determine the rotation angle ω_{opt} that optimizes jointly all four compatibility scores, the straightforward approach is to fuse them linearly in the objective function. To accomplish this it is important to note that only the two compatibility scores ${}^1m_e(\omega)$ (length of adjacent contours) and ${}^4m_e(\omega)$ (classification results) are subject to maximization, while the other two scores ${}^2m_e(\omega)$ and ${}^3m_e(\omega)$ (overlap between pieces) need to be minimized.

Since our compatibility scores yield values on different scales, we have to normalize them individually to a common range of values. For each score ${}^im_e(\omega)$ we map the range of values to $[0, 1]$ by subtracting the minimum $\underline{v}_i$ and dividing by the reciprocal of the maximum score minus the minimum score, i.e., $(\bar{v}_i - \underline{v}_i)^{-1}$. That is, the *normalized compatibility scores* used in our objective function are obtained by:

$$ {}^i\bar{m}_e(\omega) = (\bar{v}_i - \underline{v}_i)^{-1}({}^im_e(\omega) - \underline{v}_i) \tag{6.6} $$

To combine these normalized scores, we use a weight vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_4)$ that contains one weight per score and is constrained to satisfy $\sum_i \lambda_i = 1$. The objective function is then defined by

$$ f_e(\omega; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}) = \boldsymbol{\lambda}^T \bar{\boldsymbol{m}}_e(\omega), \tag{6.7} $$

where

$$ \bar{\boldsymbol{m}}_e(\omega) = \left({}^1\bar{m}_e(\omega), 1 - {}^2\bar{m}_e(\omega), 1 - {}^3\bar{m}_e(\omega), {}^4\bar{m}_e(\omega)\right) \tag{6.8} $$

$$ \underline{\boldsymbol{v}} = \left[\underline{v}_i\right]_{i=1,\ldots,4} \quad \textit{(minima)} \tag{6.9} $$

$$ \bar{\boldsymbol{v}} = \left[\bar{v}_i\right]_{i=1,\ldots,4} \quad \textit{(maxima)} \tag{6.10} $$

Minima $\underline{\boldsymbol{v}}$ and maxima $\bar{\boldsymbol{v}}$ are updated throughout the hill-climbing optimization whenever new spatial configurations are examined. A more thorough explanation on

Algorithm 6.1: Single-scale hill-climbing (`hc-sscale`)

Input : Current estimate $\hat{\omega}$
Quantifiers : Piece indices (k, l), matching candidate (i, j)
Parameters : Step width α, max. number of steps n
Output : Refined estimate $\hat{\omega}$, normalization constants $\underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}$

Initialization
  $e = (k, l, i, j)$
  $m = 0, \Omega = \emptyset$

Perform single-scale search
  while $m < n$ **do**
    `/* define grid points */`
    $\Omega = \Omega \cup \{\hat{\omega} - \alpha, \hat{\omega}, \hat{\omega} + \alpha\}$
    `/* compute scores and normalization constants */`
    foreach *type of compatibility score* i **do**
      $(\underline{v}_i, \bar{v}_i) = \left(\min_{\omega \in \Omega} {}^{i}m_e(\omega), \max_{\omega \in \Omega} {}^{i}m_e(\omega)\right)$
    `/* examine nearby grid points */`
    $\beta = \operatorname{argmax}_{\gamma \in \{-\alpha, 0, +\alpha\}} f_e(\hat{\omega} + \gamma; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}})$
    `/* stop if no improvement was made */`
    if *not* $(f_e(\hat{\omega} + \beta; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}) > f_e(\hat{\omega}; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}))$ **then**
      break
    `/* loop variables */`
    $\hat{\omega} = \hat{\omega} + \beta$
    $m = m + 1$

return $(\hat{\omega}, \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}})$

how this adaptive re-normalization works is provided in algorithm `hc-sscale` and `hc-mscale`, which are presented next.

6.3.3 Coarse-To-Fine Grid Search

Hill-climbing eliminates the need for exhaustively examining all possible solutions by performing a local search over the solution space. Despite the fact that we consider only one dimension, evaluating the compatibility scores for each $\omega \in \Omega$ at a fine resolution would still be too time consuming. Instead, we perform a coarse-to-fine grid search. The algorithm for the local search on a single-scale is presented in procedure `hc-sscale`, and the mutli-scale grid search is summarized in procedure `hc-mscale`.

Algorithm 6.2: Coarse-to-fine multi-scale hill-climbing (`hc-mscale`)

Input : Pieces $\mathcal{P}_k$, $\mathcal{P}_l$
Quantifiers : Piece indices (k, l), matching candidate (i, j)
Parameters : Step width α, adaption rate η, max. number of steps n, max. number of grid resolutions s_{max}
Output : Best estimate ω_{opt}, normalization constants $\underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}$

Initialization
  $e = (k, l, i, j)$
  $\hat{\omega} \leftarrow$ *compute initial estimate for pieces* $\mathcal{P}_k$ and $\mathcal{P}_l$ (see section 6.2)
  $s = 0$
  $\omega_{opt} = \hat{\omega}$

Perform multi-scale search
  while $s < s_{max}$ **do**
    `/* perform search on current scale */`
    $(\hat{\omega}, \underline{\boldsymbol{v}}^{(s)}, \bar{\boldsymbol{v}}^{(s)}) = \texttt{hc-sscale}_e(\hat{\omega}; \eta^s \alpha, n)$
    `/* subsume constants from all scales */`
    foreach *type of compatibility score* i **do**
      $(\underline{v}_i, \bar{v}_i) = (\min_{s' \leq s} \underline{v}_i^{(s')}, \max_{s' \leq s} \bar{v}_i^{(s')})$
    `/* no better solution was found */`
    if *not* $(f_e(\hat{\omega}; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}) > f_e(\omega_{opt}, \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}))$ **then**
      break
    `/* store current best solution */`
    $\omega_{opt} = \hat{\omega}$
    `/* continue search on finer scale */`
    $s = s + 1$

return $(\omega_{opt}, \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}})$

The latter method starts on the coarsest grid with an initial estimate $\hat{\omega}$, which is associated with the initial rigid transformation $H_e(\hat{\omega})$. Upon termination of the algorithm we obtain a refined angle ω_{opt} that defines the final transformation $H_e(\omega_{opt})$. For the sake of brevity, the algorithms presented here do not implement a caching strategy for already visited grid points. In practice, redundant computations can easily be avoided by storing compatibility scores. Computing objective function values for known grid points then only requires a re-normalization of compatibility scores rather than evaluating them from scratch.

We empirically set the initial step size to $\alpha = 5°$ and the adaption rate to $\eta = 0.5$.

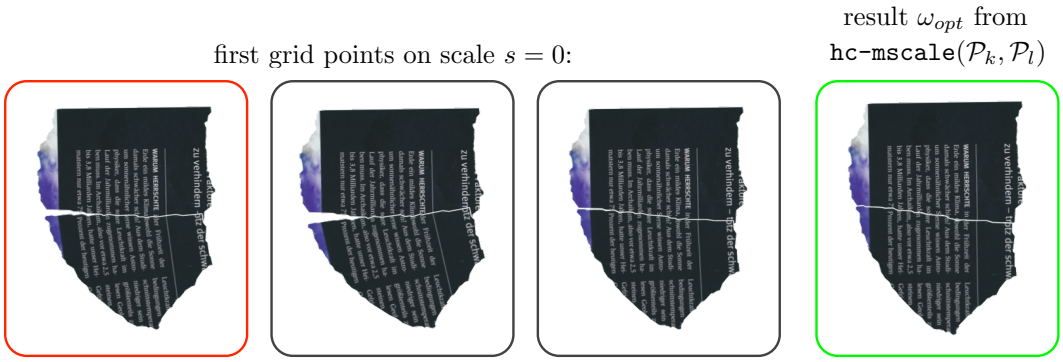


Figure 6.2: Examples for different spatial configurations associated with grid points examined throughout the hill-climbing optimization. **From left to right:** Aligned version $H_e(\hat{\omega} + \gamma)\langle\mathcal{P}_l\rangle$ of the second piece regarding the (i) initial estimate $\hat{\omega}$ ($\gamma = 0°$), (ii) immediate grid neighbor at $\gamma = +5°$ and (iii) $\gamma = -5°$. The rightmost configuration on the finest grid resolution is the optimum $\omega_{opt} = \hat{\omega} - 6.875°$.

The multi-scale search operates on at most $s_{max} = 5$ grid resolutions, corresponding to a resolution on the finest grid to $\eta^{s_{max}-1}\alpha = 0.3125$ degrees. Every single-scale search is set to examine at most $n = 4$ grid points other than the current estimate $\hat{\omega}$, towards either side of that value. Thus, given the above parameters we can update the rotation angle within interval $\hat{\omega} \pm 20°$ on the coarsest scale. In preliminary experiments we found out that this is sufficient to obtain accurate results as the initial guess already provides a decent estimate.

Examples for spatial configurations which are obtained from running the multi-scale search are given in figure 6.2. As can be seen on the left, the spatial configuration entailed by the initial estimate $\hat{\omega}$ is not very precise. The middle two configurations correspond to neighboring grid points of $\hat{\omega}$ on the coarsest resolution (scale $s = 0$). We note that the objective function values are seemingly strongly correlated with our perception of how well the pieces are aligned. Regarding the ground truth, the degree of misalignment between two pieces is evaluated in terms of "adjustment cost". In the figure these values Ξ, which are formally introduced in section 6.4.3, are reported underneath each configuration. We will use these cost values later for our quantitative evaluation.

The four spatial configurations shown in figure 6.2 correspond to a subset of the grid points examined throughout the hill-climbing optimization. The plot in figure 6.3 shows all grid points that were evaluated during the coarse-to-fine search. The series of plots shows the four normalized compatibility scores that are subject to maximiza-

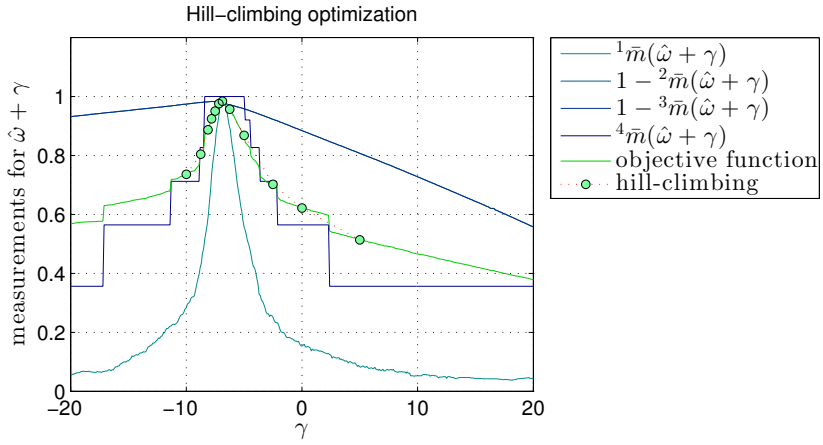


Figure 6.3: Hill-climbing optimization exemplified for an adjacent piece-pair. The coarse-to-fine grid search starts from an initial estimate $\hat{\omega}$ and offset $\gamma = 0°$. It incrementally updates this offset until no further improvement can be made. Note that the compatibility scores have been evaluated densely only for sake of visualization.

tion, as well as their linear combination in form of the objective function. Note that the two scores for the relative and the absolute intersection superimpose each other. This is only the case for a single piece-pair. Considering multiple piece-pairs during reconstruction disambiguates the two scores. For this experiment we assigned equal weights to all scores by setting $\lambda_i = 0.25$ for $i = 1, \ldots, 4$. Other weighting schemes are experimentally evaluated in section 6.4.

6.3.4 Choosing the Best Spatial Configuration

Up to this point, we have discussed how an initial estimate is obtained for any given inlier candidate and how to this estimate can be refined with a local search thereafter. We run a multi-scale search for each candidate obtained from algorithm `sig-verify`, which gives us multiple spatial configurations. Obviously, we do not know with absolute certainty whether a given inlier candidate has been classified correctly. Therefore, some spatial configurations are computed from inliers and some from outliers.

The key observation is that incorrect configurations obtained from outliers typically yield lower objective function values. Thus, we choose the spatial configuration with the highest objective function value. It must be emphasized that individual runs of algorithm `hc-mscale` (for different inlier candidates) yield different normalization constants. We account for this circumstance by choosing the best transformation only once after re-normalizing all objective function values with the final normalization constants.

One can proceed in similar fashion when having to process multiple piece-pairs. The idea of re-normalization is not only employed for our evaluation in section 6.4 but also for our reconstruction algorithm presented in the next chapter (see algorithm `kruskal-binary`). In order to deal with multiple piece-pairs properly, our evaluation methodology anticipates a simple graph initialization procedure, which formally will not be introduced until the next chapter. In the next section we briefly discuss all of the procedure's important aspects that are relevant for the understanding of our evaluation.

6.4 Evaluation

In this section we evaluate our proposed hill-climbing method. To this end, we first randomly rotate pieces in place before computing the presumably optimal transformation to align each piece-pair. The correctness of these results is then evaluated with respect to the ground truth, i.e., by assessing how much the entailed spatial configuration deviates from the pieces' correct relative positioning.

6.4.1 Methodology

The methodology for the evaluation of hill-climbing results is as follows:

Collecting Examples. First we use the ground truth to determine the belonging of pieces to pages within {test} of the *bdw082010* dataset. Each of those 48 pages is evaluated separately.

Discovering Spatial Configurations and Normalization Constants. Each piece is rotated randomly once in advance to ensure that our method is unbiased regarding the orientation of scanned pieces. We then determine the best spatial configuration *separately* for any given piece-pair and any inlier candidate obtained from `sig-verify`. We run a coarse-to-fine hill-climbing for each such point-pair which gives us *multiple* configurations for each piece-pair. In the last step we subsume all normalization constants and perform a re-normalization of all objective function values.

Storing Meta-Information in the Document Graph. The aforementioned steps are summarized in algorithm `graph-init-binary`, which is one part of the reconstruction algorithm discussed in the next chapter. The algorithm constructs a "document graph" $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_{\mathcal{V}}, l_{\mathcal{E}})$ for pieces of a single page and stores the outcome of each hill-climbing optimization. Each of the nodes in $\mathcal{V}$ represents one piece, and the edges in $\mathcal{E}$ represent the different spatial configurations between a given pair of pieces. Since

there are typically multiple edges connecting two nodes, we distinguish them by writing $e = (k, l, i, j) \in \mathcal{E}$, which refers to one particular spatial configuration of piece-pair $(\mathcal{P}_k, \mathcal{P}_l)$ obtained from inlier candidate (i, j). Three types of meta-information are relevant here:

- Σ_Z stores the random transformation applied to each piece.
- Σ_H associates an edge with the transformation obtained through hill-climbing.
- Σ_W maps each edge to the objective function value obtained for its transformation (after re-normalization with the final normalization constants).

The evaluation described so far is summarized in lines 6–7 of procedure `hc-eval`.

Choosing the Best Configuration for each Piece-Pair. Our evaluation concentrates on piece-pairs with four or more inliers, which we earlier dubbed the *strongly adjacent* pieces (see section 3.8). Non-adjacent pieces can be disregarded because they have no inliers and thus provide no insight as to whether or not hill-climbing worked correctly.

For each pair of strongly adjacent pieces we choose the most promising spatial configuration, i.e., the edge having the highest objective function value in Σ_W (line 10). Next we determine the transformation matrices to position both pieces (line 11) adjacent to each other: Z_k is set to the transformation matrix that was used during the graph initialization to randomize the orientation of piece $\mathcal{P}_k$. Likewise we proceed for the transformation Z_l of the second piece. However, there we also take the hill-climbing transformation $\Sigma_H[e^*]$ into account. In the last step we determine how well the two aligned pieces $Z_k\langle\mathcal{P}_k\rangle$ and $Z_l\langle\mathcal{P}_l\rangle$ resemble their relative positioning in the manually reconstructed page. For this purpose, we compute a rigid transformation that corrects any potential misalignment (line 12) of the two pieces. The parameters $\varsigma(Z_k, Z_l)$ of this transformation provide the basis for the computation of the adjustment cost Ξ, which is discussed next.

6.4.2 Rearranging Misaligned Piece-Pairs based on Ground Truth

Let Z_k and Z_l be the two transformations that position pieces $\mathcal{P}_k$ and $\mathcal{P}_l$ in the plane. If both of these transformations were correct, $Z_k\langle\mathcal{P}_k\rangle$ and $Z_l\langle\mathcal{P}_l\rangle$ would be the repositioned version of those pieces whose *relative* spatial configuration is the same as specified in the ground truth. In practice, however, the pieces' predicted spatial arrangement may deviate from the correct solution. We now go into detail and explain how this deviation can be measured quantitatively.

Algorithm 6.3: Hill-Climbing evaluation (`hc-eval`)

Input : Pieces of all M pages within {test} of the *bdw082010* dataset
Output : List of adjustment costs

Initialization
  $L = \emptyset$
  set classification threshold D to obtain a recall of 0.9

Evaluation of hill-climbing results
  foreach *page in the test set* **do**
    `/* retrieve pieces from current page */`
    $\{\mathcal{P}_1, \ldots, \mathcal{P}_N\} \leftarrow \text{Page}_i$
    `/* run hill-climbing on every piece-pair */`
    $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E}) \leftarrow$ `graph-init-binary`$(\mathcal{P}_1, \ldots, \mathcal{P}_N; D)$
    `/* consider each (unique) piece-pair */`
    foreach $k < l$ **do**
      if *piece-pair* $(\mathcal{P}_k, \mathcal{P}_l)$ *is strongly adjacent* **then**
        `/* choose best spatial configuration */`
        $e^* = \underset{i,j\,:\,e=(k,l,i,j)\in\mathcal{E}}{\operatorname{argmax}} \Sigma_W[e]$
        `/* transformations for aligned piece-pair */`
        $Z_k = \Sigma_Z[k]$, $Z_l = \Sigma_H[e^*]\Sigma_Z[l]$
        `/* compute adjustment cost and add to list */`
        $L = L \cup \{\Xi(\varsigma(Z_k, Z_l))\}$
  `/* sort cost values in ascending order */`
  `sort`(L)

return L

Recall from section 3.8 that for each inlier (i, j) in the set of inliers S_{kl} we can compute a residual vector $\boldsymbol{\epsilon}_{ij}^{kl}$ (see equation (3.10)):

$$\boldsymbol{\epsilon}_{ij}^{kl} = Z_k(G_k^{-1} G_l \hat{\boldsymbol{s}}_j^l) - Z_l \hat{\boldsymbol{s}}_j^l \tag{6.11}$$

Regarding the first point i, $\boldsymbol{\epsilon}_{ij}^{kl}$ specifies the offset between the true position of point j (right term) and its *expected position* (left term), where G_k and G_l are the pieces' transformations stored in the ground truth.

Based on inliers in $\mathcal{S}_{kl}$ we now quantify the degree of misalignment entailed by Z_k and Z_l. Therefore, we determine a rigid transformation that repositions one piece

to correct the misalignment. Without loss of generality, let $\mathcal{P}_k$ be the piece that remains unmodified in the process, while the other piece is getting repositioned. We first compute the shortest translation from all residual vectors of inlier $(i, j) \in S_{kl}$:

$$(i^*, j^*) = \underset{(i,j)\in\mathcal{S}_{kl}}{\operatorname{argmin}} ||\boldsymbol{\epsilon}_{ij}^{kl}||^2 \tag{6.12}$$

Accordingly, the shortest translation vector corresponds to $\boldsymbol{t}^* = \boldsymbol{\epsilon}_{i^*j^*}^{kl}$. Afterwards, we treat the second piece as if it was shifted by offset $\boldsymbol{t}^*$ and determine the optimal rotation angle α^* in order to superimpose all inliers. We rotate the repositioned piece around rotation center $\boldsymbol{r} = Z_l\hat{\boldsymbol{s}}_{j^*}^l + \boldsymbol{t}^* = Z_k(\hat{\boldsymbol{s}}_{i^*}^k + \boldsymbol{d}_{i^*j^*}^{kl})$. This rotation center corresponds to the expected position of point j^* regarding i^*, in the coordinate system of piece $Z_k\langle\mathcal{P}_k\rangle$. Using rotation matrices $R(\alpha)$ that perform a rotation around the origin by α leads to the following minimization problem:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \sum_{(i,j)\in\mathcal{S}_{kl}} ||Z_k(\hat{\boldsymbol{s}}_i^k + \boldsymbol{d}_{ij}^{kl}) - T(\boldsymbol{r})R(\alpha)T(-\boldsymbol{r})(Z_l\hat{\boldsymbol{s}}_j^l + \boldsymbol{t}^*))||^2 \tag{6.13}$$

We call $\varsigma(Z_k, Z_l) = (\boldsymbol{t}^*, \alpha^*)$ the *residual parameters* entailed by Z_k and Z_l. Note that applying this translation and rotation to $Z_l\langle\mathcal{P}_l\rangle$ recovers the correct relative spatial configuration as specified in the ground truth.

6.4.3 Adjustment Cost

We quantify the correctness of alignments by computing *adjustment cost* for each piece-pair. As discussed in the previous section, we determine the residual parameters of a rigid transformation that corrects any potential misalignment. Based on these parameters we then assess costs in two steps. First, the cost for the residual translation is computed by

$$\xi(\boldsymbol{t}^*) = 1 - e^{-\eta||\boldsymbol{t}^*||}, \tag{6.14}$$

where $\eta = 250$ (in pixel) is used for normalization. Note that this value has been set empirically to obtain meaningful values for pieces digitized with a scan resolution of 200 dpi. The second cost for the residual rotation is computed by

$$\xi(\alpha^*) = \frac{1}{180}|\alpha^*|, \tag{6.15}$$

which takes on values from $[0, 1]$ since $\alpha^* \in (-180, 180]$ (in degrees). If the two pieces $Z_k\langle\mathcal{P}_k\rangle$ and $Z_l\langle\mathcal{P}_l\rangle$ are arranged as in the ground truth, both costs equal 0. The *adjust-*

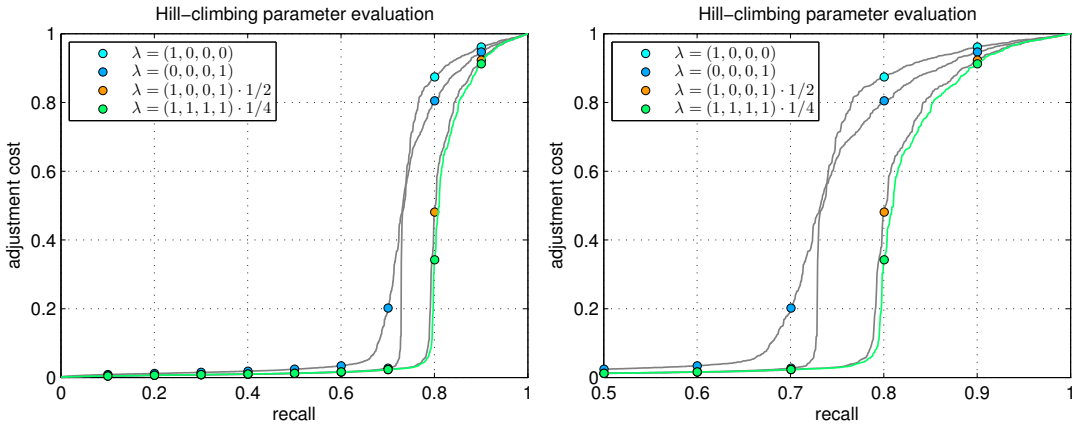


Figure 6.4: Comparison of different parametrizations for our coarse-to-fine hill-climbing in terms of adjustment cost (see eq. (6.16)). The combination of all compatibility scores results in the lowest cost at any given level of recall. The right plot shows the same data in a smaller region of interest, for recall levels of 0.5 to 1.0. See text for details.

ment costs are then obtained by fusing the two quantities as follows:

$$\Xi(\varsigma(Z_k, Z_l)) = \xi(\boldsymbol{t}^*) + \big[1 - \xi(\boldsymbol{t}^*)\big]\xi(\alpha^*) \tag{6.16}$$

Intuitively, if the residual offset vector is $\boldsymbol{t}^* = \mathbf{0}$, pieces are known to be adjacent for at least one inlier. In this case the only source of error is an incorrect orientation of pieces, which is reflected by the fact that $\Xi(\varsigma(Z_k, Z_l)) = \xi(\alpha^*)$ holds. On the contrary, if the pieces also need to be translated ($\boldsymbol{t}^* \neq \mathbf{0}$), more emphasis is put on the cost for correcting the translation rather than the orientation. That is, we reduce the cost for the orientation by a factor $\big[1 - \xi(\boldsymbol{t}^*)\big]$. This weighting scheme accounts for the fact that the orientation of pieces conveys little information about the degree of misalignment if they are attached at an outlier.

6.4.4 Experiments

In our experiments we evaluate the impact of different linear weighting schemes $\boldsymbol{\lambda}$ used in the hill-climbing objective function. Recall that this vector determines how the compatibility scores are combined into a single value. To find the best $\boldsymbol{\lambda}$ we use all strongly adjacent piece-pairs from {test} and compute their adjustment cost values using procedure `hc-eval`. Since lower adjustment cost correspond to more accurately aligned pieces, this allows us to choose the optimal linear weights.

The results are shown in figure 6.4. Here we plot the adjustment cost as a function

of recall, i.e., the percentage of strongly adjacent piece-pairs. From the plots it becomes apparent that using a uniform weighting scheme allows us to identify the correct spatial configuration for almost 80% of all piece-pairs, with negligibly low adjustment cost. It can also be seen that using only scores regarding the length of adjacent contours ${}^{1}\bar{m}_e(w)$ (i.e., $\lambda = (1, 0, 0, 0)$) performs very similarly to using only scores based on inlier candidates ${}^{4}\bar{m}_e(w)$ (i.e., $\lambda = (0, 0, 0, 1)$). However, the combination of both yields substantially lower adjustment cost for any given level of recall. Interestingly, adding the two intersection criteria ($1 - {}^{2}\bar{m}_e(w)$ and $1 - {}^{3}\bar{m}_e(w)$) only results in a very moderate performance improvement. We conjecture that this is because the initial estimates for the hill-climbing procedure are almost always precise up to a few degrees. Therefore, content overlap between pieces rarely happens, which is why the objective function is mostly unaffected by adding those two scores.

6.5 Summary

In this chapter we have presented a coarse-to-fine hill-climbing approach for the alignment of piece-pairs. Our method relies on geometric and content-based compatibility scores to assess the quality of spatial configurations of pieces. We have argued that these scores need to be normalized individually in order to be useful for the objective function that is to be optimized. To this end, we have shown how an adaptive normalization scheme can be employed without being susceptible to systematic changes in characteristics of the input data. Instead of relying on normalization constants that are chosen empirically, we choose a purely data-driven approach that continually updates these constants as new spatial configurations are examined.

Chapter 7

Agglomerative Reconstruction of Individual Pages

7.1 Motivation

It has been shown by Demaine and Demaine [13] that a general instance of the jigsaw puzzle problem is NP-complete. In spite of pieces having fixed orientations, which in the case of square pieces effectively narrows down the number of piece-combinations to only four, this problem remains hard to solve. The main reason why these puzzles represent a challenge is that pieces cannot be matched unambiguously. Not very surprisingly, the same property that makes for an interesting puzzle also makes it computationally demanding. More precisely, the problem is that a partial solution need not necessarily be correct even if two pieces fit together locally along their matching boundary. Thus, if the puzzle depicts a natural image, we can draw the conclusion that a greedy reconstruction approach is destined to fail whenever local features are not sufficient for an unambiguous distinction of correct from incorrect matches.

In contrast to jigsaw puzzles, the reconstruction of hand-torn documents involves pieces of completely arbitrary orientation. From a purely combinatorial perspective, not knowing the pieces' orientations tremendously complicates the problem: Instead of only four possible spatial configurations, pieces can be aligned in virtually infinitely many ways. We alleviated this problem in the previous two chapters, where we have shown that the number of spatial configurations to be considered can significantly be reduced. We first identified a set of inlier candidates (see chapter 5), which were then used in our hill-climbing method (see chapter 6) to identify the best alignment of pieces. As has been shown experimentally in the last chapter, this procedure allows us

to recover the spatial configuration of up to 80% of the piece-pairs with high accuracy.

In spite of these real-world issues, document pieces also offer an advantage over square jigsaw pieces: Many of our real-world puzzle pieces have a characteristic shape, which provides valuable information about the compatibility of pieces from a geometric perspective. We made use of this additional source of information in our objective function of our hill-climbing method. Besides, comparing objective function values for different piece-pairs allows us to induce a ranking among them. This effectively takes away most of the problem's combinatorial character, provided that more accurately aligned piece-pairs are ranked higher than misaligned pieces.

To make this more clear, we draw a comparison to solving edge-matching puzzles. As discussed earlier in section 1.2, pieces in this kind of puzzle are typically identically shaped but differently patterned. Thus, the matching of pieces is inherently ambiguous as only a fixed set of predefined colors is used for their patterns. Accordingly, for any given two pieces one can only decide whether they (i) do *not* fit (different colors along the edges), or else they (ii) do *possibly* fit (equal colors). That is, whether or not two pieces with the same color belong together cannot be decided without additional contextual knowledge. Since pieces cannot be distinguished by other means than their patterns, an approach for solving such a puzzle needs to account for its inherent combinatorial nature, e.g., by using a satisfiability solver as in [28]. In contrast to this type of artificial puzzle, the fact that we can always induce a ranking among aligned piece-pairs led us to believe that we can solve our real-world problem in a greedy manner.

Early variants of our reconstruction algorithm were presented in [42, 43]. In the former work we also introduced the performance measure used here for our quantitative evaluation. The organization of this chapter is as follows: We discuss a complete approach for the reconstruction of an individual document page given its set of pieces. To this end, the algorithm introduced in section 7.2 performs an agglomerative (i.e., a bottom-up) reconstruction of the document. We iteratively merge the most promising pieces and demonstrate how evidence from this merging step can be incorporated into the graph's link structure. Afterwards, in section 7.3 we conduct experiments to quantitatively evaluate the algorithm's reconstruction performance. Finally, we conclude the chapter with some remarks in section 7.4.

7.2 Graph-Based Reconstruction Algorithm

We now introduce an implementation of Kruskal's algorithm [35] tailored specifically to our application. In its original form, the purpose of this graph-based algorithm is

to find a *minimum spanning tree* of a given *weighted* graph. If this underlying graph is connected, the algorithm outputs a set of edges that satisfies two conditions: (i) *Spanning tree*: the set forms a tree that connects all the nodes in the graph. (ii) *Minimality*: the sum of edge weights is minimal. If the graph is not connected, the algorithm determines a minimum spanning tree of each of its connected components. In this case, the resulting set represents a minimum spanning *forest*.

Based on the resulting set of spanning tree edges, our algorithm also computes one rigid transformation for each piece and outputs this set of transformations. Each such transformation defines the position of a piece in the assembled document, satisfying two requirements: (i) Pieces should not overlap each other. (ii) Each piece needs to be adjacent with (at least) one other piece, and those pairwise adjacent pieces should be compatible in terms of their image content along their adjacent boundaries.

7.2.1 Modifications to Kruskal's Algorithm

To accomplish the abovementioned goals, we use an underlying graph in which each piece is represented by a node, and edges between two nodes are associated with spatial configurations. Since pieces are aligned through hill-climbing (see chapter 6), it is an obvious choice to use the objective function values for the graph's edge weights. Recall that these values have been normalized to range $[0, 1]$. Since larger values indicate more accurate alignments, we compute a *maximum spanning tree* instead of picking edges with total *minimal* weight.

To make Kruskal's algorithm applicable to this graph, two modifications have to be made, which are sketched here very briefly. A more thorough explanation is provided in the subsequent sections.

First of all, to model the fact that there are multiple spatial configurations of each piece-pair, we make use of a *multigraph* in which two nodes can be connected by more than one edge. We deliberately choose not to resolve this ambiguity in advance during the graph construction. The reasoning behind this is the following: Our algorithm proceeds in an iterative fashion by merging nodes into clusters of nodes, i.e., individual pieces are combined into groups of aligned pieces. Every merging step provides additional information about the spatial relationship of a subset of pieces, allowing us to invalidate potentially incorrect spatial configurations. This leads to more robust decisions throughout later iterations.

This first change directly relates to our second modification of the algorithm: We propose to dynamically update both the edge weights and the link structure of the graph to account for new evidence. Whenever pieces are merged into a new group of

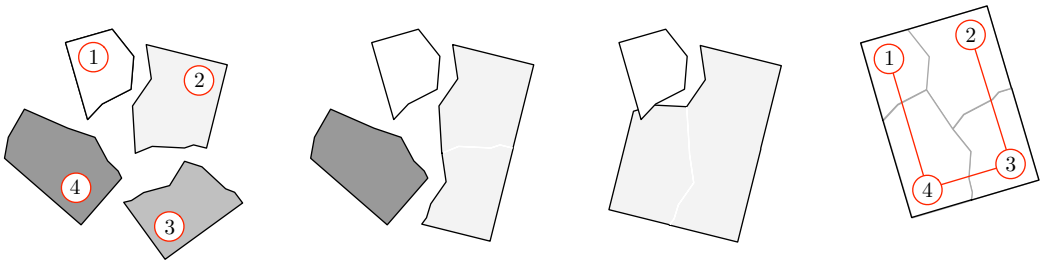


Figure 7.1: Illustrating example of how four pieces (iteration 0) are assembled into a sound solution (iteration 3), which is represented in terms of spanning tree edges (highlighted in red color). The edges of the underlying document graph become updated once after each merging step to correctly reflect the adjacency relationship among all adjacent pieces in the solution. See text for details.

aligned pieces, we once update the graph before the next merge. By incorporating additional information into the graph we manage to make only well-informed decisions that in the end lead to an accurately reconstructed document. Note that independently of our research a very similar idea was put to practice in [51].

We want to emphasize that without the repeated graph updates, the spanning tree would not correctly reflect our optimization goal that *all* pairwise adjacent pieces have to be compatible along their adjacent boundaries. In order to facilitate the understanding of this circumstance, figure 7.1 provides an illustrating example: Starting from four pieces in their initially random positions, the algorithm first chooses the most promising piece-pair (iteration 1). After piece 2 and piece 3 are merged into one new "artificial piece", the hill-climbing procedure is applied to each of the remaining two pieces in order to align them with this new partial solution. Since we update the graph according to the new hill-climbing outcomes, our next merging decision (iteration 2) does now reflect the fact that, when having to position piece 4 correctly, it will not only be adjacent to piece 3 but also to piece 2. The same principle applies to the last merging step (iteration 3). In summary, the sum of the edge weights of the resulting spanning tree (shown on the righthand side) comprises the compatibility scores of *all* adjacent pieces in the solution.

7.2.2 Representing Spatial Configurations in the Document Graph

All spatial configurations of pieces are represented in a *document graph*, which is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_{\mathcal{V}}, l_{\mathcal{E}})$. By set $\mathcal{V} = \{1, \ldots, N\}$ we refer to the set of *nodes* in the

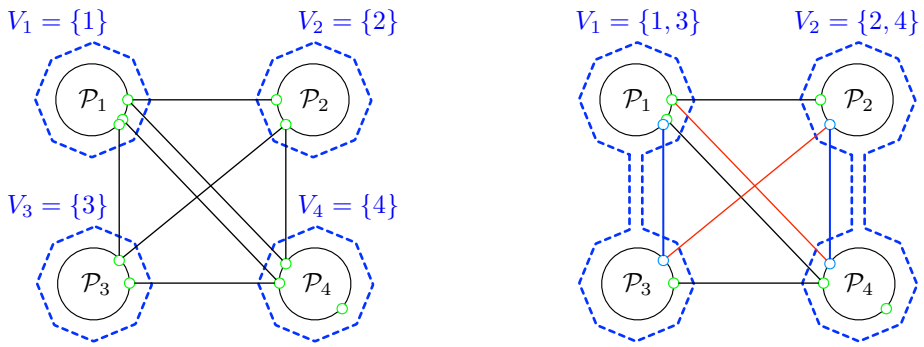


Figure 7.2: Illustrating example for a document graph with only four nodes. **Left**: Graph after initialization. Each node has support points "nested" inside, allowing multiple edges between two nodes. **Right:** Graph after two merging steps that combined two pieces each into separate clusters. Once pieces are combined into clusters, "intra-cluster" edges (blue) no longer need to be considered. Besides, some "inter-cluster" edges (red) can also be invalidated from contextual knowledge (see text for details).

graph, each of which representing a single piece. That is, the k-th node represents the piece $\mathcal{P}_k$. The *edges* $e \in \mathcal{E}$ are identified by quadruples $e = (k, l, i, j)$, which connect two nodes k and l. The latter two indices i and j of each edge e represent support points of pieces $\mathcal{P}_k$ and $\mathcal{P}_l$, respectively. The incorporation of support points in our edge representation allows us to distinguish different edges between the same pair of nodes. Allowing multiple edges in between two nodes is necessary in order to represent different spatial configurations of the nodes' respective pieces.

Additionally, the graph stores contextual information $l_{\mathcal{V}}$ as a list of rigid transformations Σ_Z. That is, by $\Sigma_Z : \mathcal{V} \to \mathbb{R}^{3\times3}$ each node $k \in \mathcal{V}$ is associated with one transformation matrix Z_k that determines the piece's position in a local image coordinate system. Besides, for each edge we store meta-information in $l_{\mathcal{E}} = (\Sigma_\Omega, \Sigma_H, \Sigma_\Upsilon, \Sigma_W)$. Since $\mathcal{G}$ establishes multiple edges between pairs of nodes that all come with contextual information, we call it a *labeled multigraph*. For edges we store the outcome of the hill-climbing optimization as well as the edge weight. First, $\Sigma_\Omega : \mathcal{E} \to (-\pi, \pi]$ represents the optimal rotation angle that is used for the alignment of pieces linked by the respective edge. Since having a fixed rotation angle eliminates the last degree of freedom of our rigid transformations, these are unambiguously defined and memorized in map $\Sigma_H : \mathcal{E} \to \mathbb{R}^{3\times3}$. Also, normalization constants from compatibility scores gathered throughout the hill-climbing procedure are stored in Σ_Υ. Finally, each edge is assigned its weight in $\Sigma_W : \mathcal{E} \to [0, 1]$, which corresponds to the objective function value of the hill-climbing procedure.

An illustrative example for a document graph is shown on the left in figure 7.2.

Algorithm 7.1: Document graph initialization (`graph-init-binary`)

Input : Pieces $\mathcal{P}_1, \ldots, \mathcal{P}_N$
Parameters : Classification threshold D
Output : Initialized document graph $\mathcal{G}$

Initialization
  Declare $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E})$ with $\mathcal{V} = \{1, \ldots, N\}$, $\mathcal{E} = \emptyset$, and
  $l_\mathcal{V} = (\Sigma_Z) = (\emptyset)$
  $l_\mathcal{E} = (\Sigma_\Omega, \Sigma_H, \Sigma_\Upsilon, \Sigma_W) = (\emptyset, \emptyset, \emptyset, \emptyset)$

Set up nodes
  `/* meta-information for nodes */`
  foreach $k = 1, \ldots, N$ **do**
    `/* set transformation for positioning randomly */`
    $\Sigma_Z[k] = T_{rand,k}$

Set up edges
  `/* consider all (unique) piece-pairs */`
  for $k < l$ **do**
    `/* obtain set of inlier candidates */`
    $A \leftarrow \texttt{sig-verify}_{k,l}(D)$
    foreach $(i, j) \in A$ **do**
      `/* add edge to edge set */`
      $\mathcal{E} = \mathcal{E} \cup \{e\}$, with $e = (k, l, i, j)$
      `/* transformations for current positions */`
      $Z_k = \Sigma_Z[k]$, $Z_l = \Sigma_Z[l]$
      `/* determine best spatial configuration */`
      $(\omega_{opt}, \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}}) = \texttt{hc-mscale}_e(Z_k\langle\mathcal{P}_k\rangle, Z_l\langle\mathcal{P}_l\rangle)$
      `/* meta-information for edges */`
      $\Sigma_\Omega[e] = \omega_{opt}$, $\Sigma_H[e] = H_e(\omega_{opt})$, $\Sigma_\Upsilon[e] = (\underline{\boldsymbol{v}}, \bar{\boldsymbol{v}})$
  `/* subsume constants from all configurations */`
  foreach *type of compatibility score* i **do**
    $(\underline{v}_i, \bar{v}_i) = (\min_{e\in\mathcal{E}} \left[\Sigma_\Upsilon[e]\right]_i, \max_{e\in\mathcal{E}} \left[\Sigma_\Upsilon[e]\right]_i)$
  `/* initialize edge weights */`
  foreach $e = (k, l, i, j) \in \mathcal{E}$ **do**
    $\Sigma_W[e] = f_e(\Sigma_\Omega[e]; \underline{\boldsymbol{v}}, \bar{\boldsymbol{v}})$

return $\mathcal{G}$

7.2.3 Graph Initialization

Consider the graph initialization procedure `graph-init-binary`. Provided with a set of pieces from one individual page, the procedure constructs the initial document graph in two steps. First, we assign a random rotation matrix $T_{rand,k}$ to each piece $\mathcal{P}_k$ (line 5). This step is necessary because all fragments were scanned in predominantly upright direction in order to facilitate the creation of our ground truth. In the second block we set up the link structure of the graph. For this, we consider each unique piece-pair once: For each pair of nodes (k, l) we determine all inlier candidates between the respective two pieces by using algorithm `sig-verify`. Each edge $e = (k, l, i, j)$ unambiguously identifies a matching candidate for the piece-pair $(\mathcal{P}_k, \mathcal{P}_l)$. Once the edge is added to the edge set, we determine the best possible spatial configuration for the two pieces at hand, using support points (i, j). To this end, we apply the coarse-to-fine hill-climbing procedure `hc-mscale`, which takes as input the randomly orientated pieces (line 12). The results are memorized in Σ_Ω, Σ_H and Σ_Υ, i.e., we store the optimal rotation angle, the transformation matrix for the alignment, as well as the normalization constants. To compute the final normalized edge weights, we have to account for the minima and maxima encountered during examining all spatial configurations (lines 14–17).

7.2.4 Kruskal's Algorithm

Next we discuss the actual reconstruction procedure `kruskal-binary`, which at its core is a spanning tree algorithm. As motivated before, the idea is to iteratively merge pieces into groups of aligned pieces. We call each such group a *cluster*. Before the first iteration, the node set is partitioned into $N = |\mathcal{V}|$ clusters, $C = \{V_1, \ldots, V_N\}$, in which each cluster is represented by a single node.

At each iteration we choose the edge having the largest weight, which corresponds to the two putatively most compatible piece-groups. More precisely, from all edges in $\mathcal{E}$ that connect two distinct clusters, we choose the maximizer $e^* = (k^*, l^*, i^*, j^*)$ with respect to the objective function values resulting from the hill-climbing optimization. By idx (introduced in line 7) we refer to a mapping that associates each node index with the index of the cluster it currently belongs to. Accordingly, $idx(k^*)$ and $idx(l^*)$ are the indices of the two clusters which are connected by the edge e^*. To avoid cluttering the algorithm we anticipate that function idx is initialized properly and updated once two clusters become merged.

An example for two clusters comprising two nodes each is depicted on the right

Algorithm 7.2: Spanning tree algorithm (`kruskal-binary`)

Input : Pieces $\mathcal{P}_1, \ldots, \mathcal{P}_N$
Parameters : Classification threshold D
Output : Set of transformations $\{Z_1, \ldots, Z_N\}$, spanning tree edges $\mathcal{E}_{span}$

1 **Initialization**
2 $\quad \mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E}) \leftarrow$ `graph-init-binary`$(\mathcal{P}_1, \ldots, \mathcal{P}_N; D)$
3 $\quad \mathcal{E}_{span} = \emptyset$
$\quad$ `/* set up clusters */`
4 $\quad C = \{V_1, \ldots, V_N\}$ with $V_k = \{k\}$
5 **Kruskal's algorithm for individual documents**
6 $\quad$ **while** $|C| > 1$ **do**
$\quad\quad$ `/* choose best transformation (if any) */`
7 $\quad\quad e^* = (k^*, l^*, i^*, j^*) = \underset{\substack{e=(k,l,i,j)\in\mathcal{E} \\ idx(k)\neq idx(l)}}{\operatorname{argmax}} \Sigma_W[e]$
$\quad\quad$ `/* stop if no transformation was found */`
8 $\quad\quad$ **if** $e^* = \emptyset$ **then**
9 $\quad\quad\quad$ break
$\quad\quad$ `/* merge nodes into a single cluster */`
10 $\quad\quad V = V_{idx(k^*)} \cup V_{idx(l^*)}$
$\quad\quad$ `/* update the pieces' positions */`
11 $\quad\quad$ **foreach** $l \in V_{idx(l^*)}$ **do**
12 $\quad\quad\quad \Sigma_Z[l] = \Sigma_H[e^*]\Sigma_Z[l]$
$\quad\quad$ `/* update remaining set of clusters */`
13 $\quad\quad C = C \setminus \{V_{idx(k^*)}, V_{idx(l^*)}\} \cup \{V\}$
$\quad\quad$ `/* add edge to spanning tree */`
14 $\quad\quad \mathcal{E}_{span} = \mathcal{E}_{span} \cup \{(k^*, l^*)\}$
$\quad\quad$ `/* update edge labels regarding new cluster */`
15 $\quad\quad \mathcal{G} =$ `update-graph-binary`$(\mathcal{G}, V, C \setminus V)$
16 **return** $\{Z_k = \Sigma_Z[k] \mid k = 1, \ldots, N\}, \mathcal{E}_{span}$

in figure 7.2. If no more edges exist between the remaining set of clusters (line 8) the procedure terminates early. In this degenerated case, the resulting edge set $\mathcal{E}_{span}$ forms a spanning forest (i.e., a set of spanning trees). For instance, this could be caused by an overly conservative choice for the classification threshold. Since $\mathcal{E}_{span}$ then represents multiple connected components, this would lead to an incomplete reconstruction with $|C|$ partial solutions left.

Algorithm 7.3: Update graph (`update-graph-binary`)

Input : Document graph $\mathcal{G}$, new cluster V, other clusters $C \backslash V$
Output : Updated graph

1 **Update edge labels**
  `/* revise edge labels associated with new cluster */`
2 **foreach** $e = (k, l, i, j) \in \mathcal{E}$ **do**
    `/* edge connects new with old cluster */`
3   **if** $V_{idx(k)} = V \; XOR \; V_{idx(l)} = V$ **then**
      `/* treat clusters as artificial pieces */`
4      $\mathcal{P}_V \longleftarrow$ `group`$(\{\Sigma_Z[k']\langle \mathcal{P}_{k'} \rangle \mid k' \in V_{idx(k)}\})$
5      $\mathcal{P}_{V'} \longleftarrow$ `group`$(\{\Sigma_Z[l']\langle \mathcal{P}_{l'} \rangle \mid l' \in V_{idx(l)}\})$
      `/* determine best spatial configuration */`
6      $(\omega_{opt}, \underline{\boldsymbol{\upsilon}}, \bar{\boldsymbol{\upsilon}}) =$ `hc-mscale`$_e(\mathcal{P}_V, \mathcal{P}_{V'})$
      `/* meta-information for edges */`
7      $\Sigma_\Omega[e] = \omega_{opt}$, $\Sigma_H[e] = H_e(\omega_{opt})$, $\Sigma_\Upsilon[e] = (\underline{\boldsymbol{\upsilon}}, \bar{\boldsymbol{\upsilon}})$
  `/* subsume constants from all configurations */`
8 **foreach** *type of compatibility score* i **do**
9    $(\underline{\upsilon}_i, \bar{\upsilon}_i) = (\min_{e \in \mathcal{E}} \left[\Sigma_\Upsilon[e]\right]_i, \max_{e \in \mathcal{E}} \left[\Sigma_\Upsilon[e]\right]_i)$
  `/* update all edge weights */`
10 **foreach** $e = (k, l, i, j) \in \mathcal{E}$ **do**
11    $\Sigma_W[e] = f_e(\Sigma_\Omega[e]; \underline{\boldsymbol{\upsilon}}, \bar{\boldsymbol{\upsilon}})$
12 **return** $\mathcal{G}$

The next step is to reposition pieces of the two selected clusters and merge their respective nodes into a single new cluster. Analogously to aligning individual pieces, we hold all pieces from one cluster fixed while applying the chosen transformation $\Sigma_H[e^*]$ to all pieces in the second cluster. Therefore, we update transformations for all pieces represented by cluster $V_{idx(l^*)}$ (line 12).

By merging two clusters into a single new piece we effectively reduce the cardinality of set C by 1, which guarantees that the algorithm terminates. The remaining set of clusters that is going to be used throughout the next iteration is obtained by:

$$C = C \setminus \{V_{idx(k^*)}, V_{idx(l^*)}\} \cup \{V\} \tag{7.1}$$

The pair of nodes (k^*, l^*) which provides the inlier candidate used for the computation of transformation $\Sigma_H[e^*]$ is then added to the set of spanning tree edges $\mathcal{E}_{span}$.

Now that pieces have been repositioned within a joint coordinate system of the partial solution (i.e., the new cluster), it is likely that decisions about the positioning of other pieces may change in light of this new evidence. For this reason, the last step is to propagate the information about the new group of pieces through the document graph and update it accordingly.

We use algorithm `update-graph-binary` to adjust the graph's link structure and edge weighs with respect to the newly created cluster V. First, we need to update edges that link nodes from any cluster in $C \backslash V$ with nodes in V. This is because having formed a bigger group of aligned pieces may lead to more accurate alignments and thereby help to invalidate previously incorrectly computed transformations. Second, we have to account for the fact that clusters accumulate more and more pieces over iterations and thus, partial solutions steadily grow bigger. To obtain meaningful objective function values in later iterations, we renormalize all edge weights to $[0, 1]$, taking into account the updated normalization constants. Note that this renormalization is conceptually the same as already used for the graph initialization.

The most important aspect about updating edges is that the aligned pieces in each cluster are represented by a single "artificial piece", as previously discussed in section 3.7. The only thing that changes in light of piece-groups is that the hill-climbing `hc-mscale`, as well as all compatibility scores (eq. (6.2)–(6.5)), are now based on two groups of aligned pieces $\mathcal{P}_V$ and $\mathcal{P}_{V'}$ instead of two individual pieces. Also note that pieces within a single cluster are always considered as a single artificial piece, hence the relative positioning of its individual pieces remains unchanged while updating the graph.

7.2.5 Heuristics for Pruning Edges

In terms of runtime the performance of our algorithm mostly depends on the number of edges that have to be considered throughout the iterations. Since edges associated with low objective function values are likely to represent incorrect spatial configurations, we apply a pruning heuristic after each iteration: After re-evaluating all edge weights, we remove any edge $e \in \mathcal{E}$ for which $\Sigma_W[e] < 0.6$ holds. Due to this simple heuristic, the link structure of the document graph becomes a lot sparser, which has a notable impact on the runtime.

A second heuristic is employed once after each merging step. The idea is to invalidate incorrect spatial configurations of any piece regarding the new cluster V. Since all pieces within the cluster are now represented as a single artificial piece $\mathcal{P}_V$, it is unnecessary to consider support points lying on the inside of that new piece. To iden-

tify these interior points, we consider each piece-pair $(k, l) \in V \times V$ and determine all support points which become mutually adjacent with each other due to the merging step. Edges incident to any of these points are removed from the graph. The reasoning behind this is that aligning other pieces with $\mathcal{P}_V$ using these support points would very likely yield a content overlap between pieces, which is why these spatial configurations can be disregarded. This circumstance is illustrated on the righthand side in figure 7.2. Edges are colored in red if at least one of their incident support points must no longer be used for the alignment of pieces.

7.3 Evaluation

Next we conduct experiments to evaluate the algorithm's effectiveness for the reconstruction of individual document pages. For this purpose we use the adjustment cost introduced earlier in section 6.4. First, we demonstrate how this performance measure can be applied to determine how accurately each reconstructed page resembles its ground truth. Furthermore, we discuss our experimental results and give qualitative examples for some reconstructed document pages.

7.3.1 mean Adjustment Cost (mAC)

From applying our reconstruction algorithm we obtain a set of edges $\mathcal{E}_{span}$, alongside a transformation Z_k for each piece $\mathcal{P}_k$ that determines the piece's final position in the reconstructed document. The number of edges depends on the problem instance and may generally vary from 0 to $N-1$, where N is the number of pieces per page. If the classifier's threshold is set too high, the document graph is not fully connected. In this case the set $\mathcal{E}_{span}$ forms a spanning forest comprising multiple spanning trees.

Based on our definition of adjustment cost in section 6.4 we define the *mean Adjustment Cost* (mAC):

$$mAC(\mathcal{E}_{span}) = (N-1)^{-1}\Big[N-1-|\mathcal{E}_{span}| + \sum_{(k,l)\in\mathcal{E}_{span}} \Xi(\varsigma(Z_k, Z_l))\Big], \tag{7.2}$$

Partly reconstructed pages are penalized by introducing a cost of 1 for each edge missing from $\mathcal{E}_{span}$. The residual parameters $\varsigma(Z_k, Z_l)$ are computed individually for each piece-pair if there is an edge connecting the two nodes in the spanning tree. In a degenerate case when our classifier's threshold is set too high, no reconstruction will take place and thus $\mathcal{E}_{span} = \emptyset$ holds. In this case we would obtain $mAC(\emptyset) = 1$.

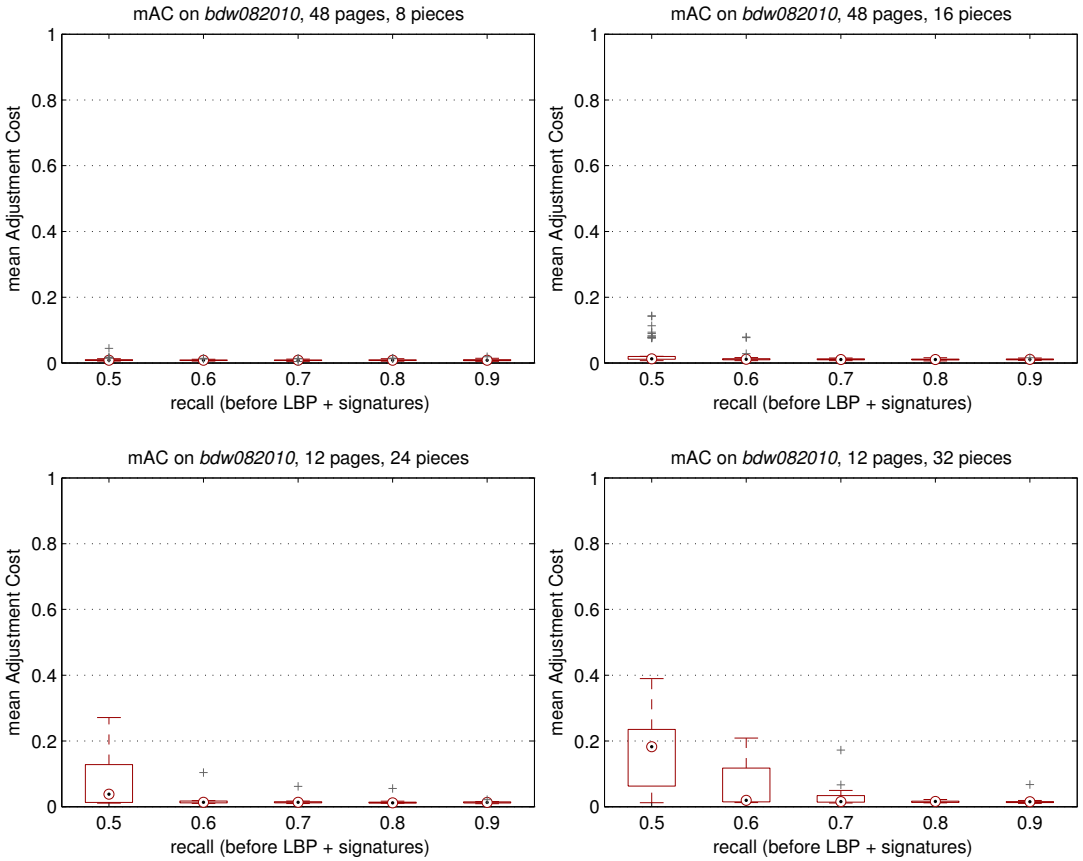


Figure 7.3: Reconstruction performance in terms of statistics on the mAC determined for different decision thresholds of AdaBoost. The plots show results for different numbers of pieces per page.

However in almost all of our experiments, $\mathcal{E}_{span}$ has $N-1$ edges and thus forms a single spanning tree. In this case, eq. (7.2) simplifies to:

$$mAC(\mathcal{E}_{span}) = (N-1)^{-1}\Big[\sum_{(k,l)\in\mathcal{E}_{span}} \Xi(\varsigma(Z_k, Z_l))\Big], \tag{7.3}$$

It now becomes apparent that $mAC(\mathcal{E}_{span}) \in [0,1]$ is simply the average over all adjustment costs. If $mAC(\mathcal{E}_{span})$ is close to 0, the reconstruction is of very high quality. The reasoning is as follows: Consider two edges from $\mathcal{E}_{span}$ which we denote by (k,l_1) and (k,l_2). If the two piece-pairs $(Z_k\langle\mathcal{P}_k\rangle, Z_{l_1}\langle\mathcal{P}_{l_1}\rangle)$ and $(Z_k\langle\mathcal{P}_k\rangle, Z_{l_2}\langle\mathcal{P}_{l_2}\rangle)$ are aligned correctly, this also entails a correct positioning of piece $Z_{l_1}\langle\mathcal{P}_{l_1}\rangle$ with regards to $Z_{l_2}\langle\mathcal{P}_{l_2}\rangle$. Since a spanning tree connects any two nodes through a chain of edges, none of the pieces in the document can be mispositioned.

7.3.2 Experiments

In our first set of experiments we separately reconstruct each of the pages from {test} of the *bdw082010* dataset. We do so once individually for each degree of fragmentation and for different classification thresholds of our AdaBoost classifier. Performance statistics regarding the pages' individual mAC values are summarized in figure 7.3. Each of the four plot groups represents the results for one fragmentation level. That is, from left to right and top to bottom we have used $N = 8, 16, 24, 32$ pieces per page, respectively[1].

We start with a short explanation on how to interpret the box plots first. The circle put in the center of each bar marks the median of all mAC values, and each bar extends to the 25th and 75th percentiles. The whiskers extend up to the most extreme points that are not yet considered as outliers, and each outlier is plotted individually (crosses)[2]. We performed reconstruction for a varying classifier threshold D, which was set to obtain recall values from 0.5 to 0.9, in steps of 0.1. Note that these recall values are computed *before* applying LBP and geometric signatures (see chapter 5). The corresponding recall values after applying procedure `sig-verify` with the number of votes set to $n_P = 1$ were reported in the plot in figure 5.5a.

Several conclusions can be drawn from this first series of experiments: First of all, it becomes apparent from figure 7.3 that lowering threshold D (higher recall) consistently improves the reconstruction results across all fragmentation levels, up to the point where the reconstruction results are essentially perfect. A lower threshold leads to more inlier candidates, which in turn means that more spatial configurations need to be considered. It should thus be made clear that better results involve higher computational costs.

A second important observation is that for higher degrees of fragmentation, the reconstruction works less reliably at low recall levels. There are two plausible explanations for this effect: First of all, if the same page has been torn into 32 instead of only 16 pieces, pieces on average have only half the size. Correspondingly, the more pieces per page, the shorter the pieces' adjacent boundaries. Since this directly influences the number of inliers per piece-pair, a fixed recall also leads to a smaller number of correctly identified inliers. A second possible explanation is that the reconstruction task naturally becomes harder the more pieces have to be considered. In particular, if many pieces need to be assembled, even small errors after each alignment step eventually

[1] Note that the *bdw082010* test set for 24 and 32 pieces only consists of a subset of 12 pages

[2] Points are drawn as outliers if their mAC is larger than $q_3 + 1.5(q_3 - q_1)$ or smaller than $q_1 - 1.5(q_3 - q_1)$, where q_1 and q_3 are the 25th and 75th percentiles, respectively. This corresponds to approximately 99.3 coverage if the points are normally distributed.

(a) 8 pieces **(b)** 16 pieces **(c)** 24 pieces **(d)** 32 pieces

(e) 8 pieces **(f)** 16 pieces **(g)** 24 pieces **(h)** 32 pieces

(i) 8 pieces **(j)** 8 pieces **(k)** 10 pieces **(l)** 10 pieces

Figure 7.4: Examples for automatically reconstructed document pages of the *bdw082010* dataset (top), as well as the *booklet* (center) and the *stieber500p* dataset (bottom row). All pages have been reconstructed using a classification threshold D corresponding to 0.9 recall (before LBP + signatures). For the result shown in 7.4j, 3 out of 7 merging decisions were incorrect, resulting in a mAC of 0.43091 (this page causes one of the three outliers in the rightmost boxplot in figure 7.6).

accumulate into larger errors. After a certain number of pieces have been merged, this may prevent us from positioning additional pieces correctly. We want to emphasize that this performance degradation is probably irrelevant for most practical applications, since assuming more than 16 pieces per page is very likely an overestimate of real problem sizes.

In the second set of experiments we finally reconstruct pages from the *booklet* and the *stieber500p* dataset, for which we plot the results in figure 7.5 and 7.6, respectively.

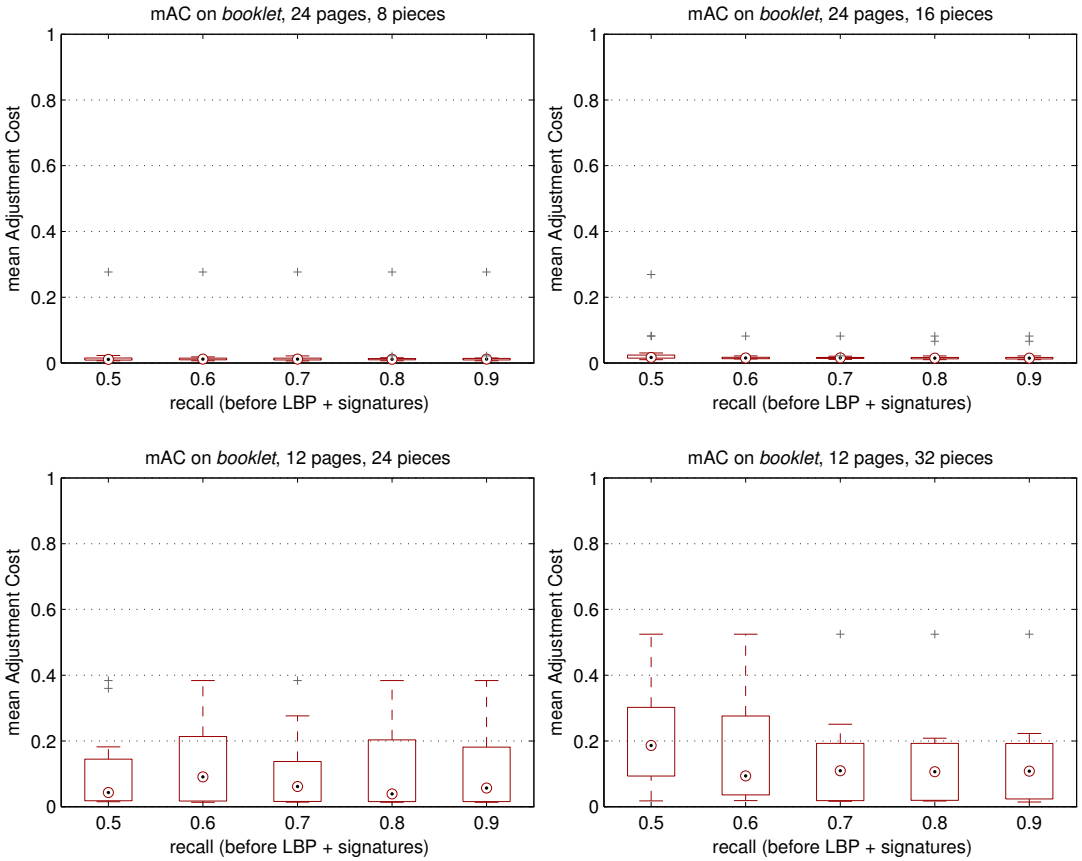


Figure 7.5: Reconstruction performance for the *booklet* dataset, using the same setup as for the plots in figure 7.3.

The key observation here is that, regardless of how much we lower the threshold D, a perfect reconstruction of all pages can not be guaranteed. The reason for this is that both datasets contain some pages with very little or even no distinguishable content, in which case it is tough to reliably match pieces in terms of content-based features.

Some examples for reconstructed pages are shown in figure 7.4. All of these pages have been reconstructed using a classification threshold D corresponding to 0.9 recall (before LBP + signatures). With the exception of the reassembled page shown in figure 7.4j, all mAC values are close to 0 as the majority of pieces has been positioned very precisely. In this rare case of failure, our algorithm made 3 incorrect merging decisions (out of 7), resulting in a mAC of 0.43091. Note that this page causes one of the three outliers in the rightmost boxplot in figure 7.6. We chose this hand-picked example to illustrate the connection between mean Adjustment Cost and reconstruction quality.

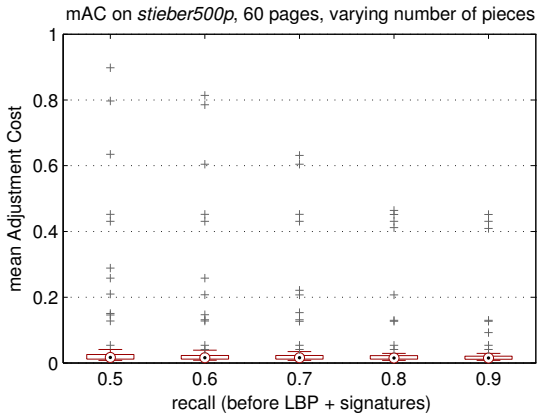


Figure 7.6: Reconstruction performance for the *stieber500p* dataset, using the same setup as for the plots in figure 7.3.

In conclusion one can say that, although some of the document pages could not be reconstructed entirely or may show a minor misplacement of pieces, our proposed method is capable of reassembling the vast majority of them almost flawlessly. One of our findings is that our algorithm almost only fails when having to deal with blank pages. However, one could argue that the reconstruction of blank pages is irrelevant in practice because we cannot get crucial information off of these pages.

7.4 Summary

In this chapter we have presented a variant of Kruskal's algorithm for the reconstruction of individual document pages. Our algorithm proceeds in an iterative manner and repositions pieces or groups of pieces until the document is reassembled. To make the original algorithm applicable to this task, two important modifications have been made: (i) To account for the different spatial configurations of two pieces (nodes), we constructed a labeled multigraph. In this graph, each configuration is represented by one edge. (ii) We resolve this inherent ambiguity step-by-step by merging pieces into groups of aligned pieces. After each iteration, we update the graph to account for the new situation. For this purpose, we realign all other pieces with the new merged piece and updated the graph's link structure and all edge weights. This leads to very robust reconstruction results because incorrect alignments can be invalidated throughout the reconstruction, once after each merging step. Finally, we have shown experimentally on three challenging datasets that our proposed method yields essentially perfect results for the majority of document pages.

Part III

Reconstruction based on Structured Output Prediction

Chapter 8

Partial Contour Matching

8.1 Motivation

The previous part of this thesis solved document reconstruction in three steps: By (i) identification of inlier candidates through binary classification and geometric verification (see chapter 5), (ii) finding a set of rigid transformations through hill-climbing (see chapter 6), and by (iii) repeated selection of the most compatible pieces in order to construct a spanning tree (see chapter 7). Although our experiments have shown that this pipeline is very effective, we feel that it can still be improved. This and the following two chapters especially focus on the efficiency of our proposed method, which makes it no longer restricted to the reconstruction of individual pages.

This chapter presents a partial contour matching approach as an efficient alternative to hill-climbing. To this end, we propose a variant of MSAC (M-estimator SAmple Consensus) [53] that determines hypotheses for recovering the correct spatial configuration of two pieces[1]. Our new approach is conceptually different from the strategy used in the first part of this thesis: We no longer require an a priori classification of all matching candidates – instead, we now directly identify inlier candidates based on contour distances along the pieces' boundaries. If we were to find (at least) two inlier candidates, such a pair directly allows us to compute a rigid transformation, without the need for a costly refinement of initial estimates through hill-climbing. For a content-based verification of different spatial configurations we later introduce a discriminative model in chapter 9. Based on this model we propose a revised reconstruction algorithm in chapter 10, which directly uses the output of this model to assess the edge weights of our document graph.

[1] Earlier variants of our method were described in previous work [44, 45].

This chapter is organized as follows: In section 8.2 we give a short introduction to RANSAC (RANdom SAmple Consensus) [17] and explain the differences to MSAC. We briefly sketch how MSAC can be used in our problem setting in order to compute hypotheses for the alignment of piece-pairs. Afterwards, we explain our algorithm for the identification of sets of inlier candidates in section 8.3. We draw a comparison to RANSAC in terms of sampling effectiveness and analyze it from a theoretical point of view. That is, we show that our approach requires less hypotheses, as opposed to a random sampling strategy. Since we obtain many different hypotheses for aligning piece-pairs, we next invalidate incorrect hypotheses. For this purpose we exploit the distance transform, a dynamic programming technique that is discussed in section 8.4. In section 8.5 we describe a sequence of rejection and selection steps that filters out incorrect hypotheses. The first rejection step is based on orientation estimates from the pieces' foreground, which are subject to verification by a discriminative model. Here we choose a support vector machine as our model, which is described in section 8.6. Afterwards, we introduce a new evaluation methodology in section 8.7, which is based on the overlap of adjacent boundary regions between aligned piece-pairs. This new performance measure is much more accessible and easier to use than the adjustment cost introduced in chapter 6. Besides its use for evaluation, it also enables us to perform non-maximum suppression on hypotheses. Thus, we can now select a diverse set of different spatial configurations per piece-pair, which substantially boosts the performance in terms of recall. Finally, we conclude the chapter with a short discussion in section 8.8.

8.2 Comparison between RANSAC and MSAC

RANSAC is a paradigm for model fitting that has become a fundamental tool in the computer vision and image processing community. In contrast to conventional techniques for parameter estimation (e.g., a least squares approach), RANSAC does not fit the model parameters using *all* of the sensed examples because the presence of noisy examples would be detrimental for the fit of the model.

Instead, RANSAC performs parameter estimation for a model in an iterative fashion. Within each iteration, it picks a random subset of data just large enough to be feasible for the estimation. It then fits a model to this set of *observations* and tests how many of the remaining data samples are compatible with the current model. Therefore, every datum is categorized into either an *inlier* or an *outlier* depending on some predefined error tolerance. The set of inliers is called the *consensus set*, and the cardi-

nality of this set is used to assess the quality of the model. RANSAC iteratively repeats model fitting for a fixed number of iterations, and, upon termination, it returns the best found model with a sufficiently high inlier count.

As discussed by Torr and Zisserman [53], the model obtained by RANSAC can be poor if the error tolerance regarding inliers is too high. If the threshold is sufficiently large, all points will be considered as inliers, hence making models undistinguishable. On the other hand, if the threshold is chosen too small, the model is unstable because adding or removing a single inlier may substantially change the model parameters. Instead of assessing the quality of a model by counting inliers, MSAC uses an estimator to assign an individual score to each inlier and a fixed penality to each outlier. It is therefore capable of disambiguating the quality of models at almost no extra computational cost, which made it the method of choice for our partial contour matching approach. Besides, consideration of RANSAC shows that counting inliers is simply a special cost function. Consequently, MSAC can be seen as a direct generalization of RANSAC. Specific to our problem, using MSAC involves the following steps:

- Based on the pieces' polygons alone we first identify a set of *inlier candidates*. To this end, we establish point-correspondences between short contour regions on the two pieces, without the need for random sampling.
- Each pair of inlier candidates allows us to compute a rigid transformation, which forms a "hypothesis" (i.e., a model) for one particular arrangement of pieces.
- The standard approach would then be to test each hypothesis regarding the set of all possible point-pairs (i.e., observations) across the two pieces. In contrast, our problem-specific variant introduces a sequence of *rejection* and *selection* criteria, which deliberately postpones using all observations until the end of the verification. This lets us identify spatial configurations of pieces very efficiently.

8.3 Identifying Candidate Sets

This section presents an algorithm that identifies inlier candidates across short contour segments. Our method exploits that support points of closed polygons are represented by cyclically ordered sets.

8.3.1 Terminology and Definitions

The idea behind the construction of candidate sets is based on a simple and intuitive observation: Given two support points i and j that form an inlier, two other support

points i' and j' are likely to be an inlier as well if the contour distances from i to i' and j to j' are similar. To clarify this point, consider the example shown in figure 8.1: If i_1 and j_1 form an inlier, one would be inclined to say that i_3 and j_3 is also an inlier.

As indicated in the figure by the short contour segments (highlighted in blue), we limit our search for inlier candidates to small parts of the closed polygonal curve. In order to define these segments, we require properties of cyclically ordered sets, which we define in the following. For this, we represent support points in $\hat{S}_k$ by an index set $I_{\hat{S}_k} = \{1, \ldots, n(k)\}$, in which $i \in I_{\hat{S}_k}$ represents the i-th support point $\hat{\boldsymbol{s}}_i^k$ on piece $\mathcal{P}_k$. We put $I = I_{\hat{S}_k}$ to avoid cluttering the notation.

Definition 1 (Cyclically ordered set)**.** Let I be a set, and let R be a ternary relation on this set, i.e., $R \subseteq I^3$ is any subset of the 3^{rd} cartesian power of I. Alternatively, we write $x \to y \to z$ for $(x, y, z) \in R$. The ternary relation R is called a *cyclic order* if it satisfies the following axioms:

1. *Cyclicity*: if $x \to y \to z$ then $y \to z \to x$
2. *Asymmetry*: if $x \to y \to z$ then $\neg(z \to y \to x)$
3. *Transitivity:* if $x \to y \to z$ and $x \to z \to z'$ then $x \to y \to z'$
4. *Totality:* if $x \neq y \neq z \neq x$ then either $x \to y \to z$ or $z \to y \to x$

If R satisfies these axioms, the pair (I, R) is called a *cyclically ordered set*. We explicitly distinguish the orientation of a cycle by writing $x \overset{\text{cw}}{\to} y \overset{\text{cw}}{\to} z$ in the clockwise case (cw) and $x \overset{\text{ccw}}{\to} y \overset{\text{ccw}}{\to} z$ in the counterclockwise case (ccw).

According to definition 1, $x \overset{\text{cw}}{\to} y \overset{\text{cw}}{\to} z$ means that while traversing support points of $\hat{S}_k$ in cw direction, starting from the support point indexed by x, we will at some point encounter the y-th point, before finally reaching the point indexed by z. Based on this ternary relation we can now define a *polygonal chain* that consists of the line segments connecting subsequent support points in between a start and an end point. For brevity we limit the discussion to the clockwise case. Definitions for the counterclockwise case can be obtained analogously.

Definition 2 (Polygonal chain)**.** Suppose we are given two indices $x, z \in I$. The set of all points encountered while traversing the polygon from x to z in clockwise direction is represented by:

$$G_{x,cw}^{z} = \{x\} \cup \{y \in I \,|\, x \overset{\text{cw}}{\to} y \overset{\text{cw}}{\to} z\} \cup \{z\} \tag{8.1}$$

This set contains all interior points $y \in I$, as well as the start and the end point.

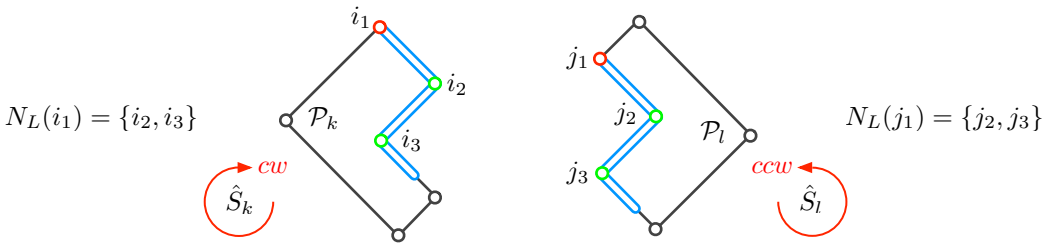


Figure 8.1: Schematic illustration of two L-neighborhoods on polygons. The maximum contour distance L with regards to starting points i_1 and j_1 are highlighted in blue. In this example, the neighborhoods comprise two support points on either piece (green circles).

It has been shown in [37] that the cyclic order R induces a *linear order* $<_{R,x}$ on $G^z_{x,cw}$ regarding the *least element* x. Formally, $i <_{R,x} i' \iff x \overset{\text{cw}}{\to} i \overset{\text{cw}}{\to} i'$ or $x = i \neq i'$. The tuple $(G^z_{x,cw}, <_{R,x})$ is a *linearly ordered set* that specifies a sequence of support points, which starts at x and ends in z.

Instead of $(G^z_{x,cw}, <_{R,x})$ we write briefly

$$x \overset{\text{cw}}{\to} z =_{df} (G^z_{x,cw}, <_{R,x}) \tag{8.2}$$

to which we refer as the *polygonal chain* from x to z. As can be seen in figure 8.1, a polygonal chain is a piecewise linear curve whose length specifies the contour distance between two points (in a given direction).

Definition 3 (Contour distance)**.** Let two support points be represented by indices $i_1, i_m \in I$. The *contour distance* in clockwise direction on piece $\mathcal{P}_k$ is the accumulated length of consecutive line segments in the polygonal chain $i_1 \overset{\text{cw}}{\to} i_m$. We define:

$$l(i_1 \overset{\text{cw}}{\to} i_m) =_{df} \sum_{n=2}^{m} ||\hat{\boldsymbol{s}}^k_{i_n} - \hat{\boldsymbol{s}}^k_{i_{n-1}}|| \tag{8.3}$$

In the degenerated case of polygonal chains with either one or zero points we set the contour distance to 0.

With regards to contour distances we can now define contour regions with maximal length L. We call these regions the L-neighborhood of a given point.

Definition 4 (L-neighborhood)**.** We say that the i'-th point is in the *L-neighborhood* of an anchor point i, on the contour of piece $\mathcal{P}_k$, if $l(i \overset{\text{cw}}{\to} i') \in (0, L]$ holds. In other words, the L-neighborhood tied to the i-th point contains only points with contour distance of at most L pixels. This circumstance is illustrated in figure 8.1. Formally we represent

the L-neighborhood as an index set:

$$N_i(L) = \{i' \in I \,|\, 0 < l(i \overset{\text{cw}}{\to} i') \leq L\} \tag{8.4}$$

Note that by definition, $i \notin N_i(L)$.

8.3.2 Finding Inlier Candidates between L-neighborhoods

According to definition 4, each L-neighborhood defines a short contour segment on the closed polygonal curve of a given piece. In the following we characterize point-correspondences between two such segments that are likely to be inliers.

Definition 5 (Candidate set)**.** Let a $(i, j) \in I_{\hat{S}_k} \times I_{\hat{S}_l}$ denote a pair of support points between two pieces. Since contour distances on both pieces are computed relative to these points, we call this tuple the *anchor points*. Based on these points we characterize our *candidate set* C_{ij} by:

$$(i', j') \in C_{ij} \implies i' \in N_i(L_1) \wedge j' \in N_j(L_1) \wedge |l(j \overset{\text{ccw}}{\to} j') - l(i \overset{\text{cw}}{\to} i')| \leq L_2, \tag{8.5}$$

By L_1 we denote the upper bound on the length of contour segments, and L_2 is the matching tolerance for a candidate to be added to the candidate set.

Before introducing our algorithm `cmp-candidate-set` for the computation of C_{ij}, let us recap the definition of candidate sets first. The set C_{ij} contains only inlier candidates (i', j'), $i' \neq i$, $j' \neq j$, from the L_1-neighborhoods of their respective anchor points. A given point-pair is added to this set only if contour distances regarding their anchor points differ by less than L_2 pixels. We note that the reverse implication generally does *not* hold. We relaxed this requirement because identifying the entire set of (many-to-many) point-correspondences would impose time requirements that are at least quadratic in the number of points in the L_1-neighborhood. In contrast, as will be discussed shortly, our algorithm only identifies one-to-many correspondences, but in return runs in linear time.

Finally, to be able to deal with the cyclic order of points in our pseudo-code algorithm, we require a function `cyclic-next`. This method traverses support points in either counterclockwise or clockwise direction. To this end, depending on the direction indicated by *cw* or *ccw* (second argument), it either increments or decrements the index of a given support point (first argument)[1].

[1] Recall from section 3.3 that the index of points increases in ccw direction.

We start with an explanation for the second piece for which points are traversed in ccw direction. Taking into account the cyclic order of points in index set $I_{\hat{S}_l}$, the next index returned by `cyclic-next` (j', ccw) is:

$$\big(j' \bmod n(l)\big) + 1 \tag{8.6}$$

The above definition maps $j' \to j' + 1$ for all integers $1 \leq j' < n(l)$, i.e., we move from the j'-th support point to its immediate successor in ccw direction. For the last support point with index $n(l)$ the function assigns $n(l) \to 1$ to complete the cycle.

For the other piece whose points are indexed by $I_{\hat{S}_k}$ we use `cyclic-next` (i', cw) to traverse points in the opposite direction:

$$\big[\big(i' - 2 + n(k)\big) \bmod n(k)\big] + 1 \tag{8.7}$$

In the cw case, the cycle is completed by jumping from $i' = 1$ to $n(k)$. All other indices $1 < i' \leq n(k)$ are simply decremented by 1.

Algorithm 8.1: Candidate set construction (`cmp-candidate-set`)

Input : Anchor points (i, j) from pieces $\mathcal{P}_k$ and $\mathcal{P}_l$
Parameters : Neighborhood range L_1, matching tolerance L_2
Output : Candidate set C_{ij}

Initialization
  $C_{ij} \leftarrow \emptyset$
  `/* extremal points of the` L_1`-neighborhoods */`
  $j_{max} = \text{argmax}_{j' \in N_j(L_1)} \left\{ l(j \overset{\text{ccw}}{\rightarrow} j') \right\}$, $i_{min} = \text{argmax}_{i' \in N_i(L_1)} \left\{ l(i \overset{\text{cw}}{\rightarrow} i') \right\}$
Add candidates inliers
  if $j \neq j_{max}$ *and* $i \neq i_{min}$ **then**
    `/* anchor points are not part of the candidate set */`
    $j' =$ `cyclic-next` (j, ccw) , $i' =$ `cyclic-next` (i, cw)
    while $true$ **do**
      `/* make a step on contour of piece` $\mathcal{P}_l$ `*/`
      if $l(j \overset{\text{ccw}}{\rightarrow} j') < l(i \overset{\text{cw}}{\rightarrow} i') - L_2$ **then**
        if $j' \neq j_{max}$ **then**
          $j' =$ `cyclic-next` (j', ccw)
        else
          $break$
      `/* make a step on contour of piece` $\mathcal{P}_k$ `*/`
      else if $l(j \overset{\text{ccw}}{\rightarrow} j') > l(i \overset{\text{cw}}{\rightarrow} i') + L_2$ **then**
        if $i' \neq i_{min}$ **then**
          $i' =$ `cyclic-next` (i', cw)
        else
          $break$
      `/* similar distances regarding anchor points */`
      else
        `/* add inlier candidate */`
        $C_{ij} = C_{ij} \cup \{(i', j')\}$
        `/* one-to-many relationship across the pieces */`
        if $j' \neq j_{max}$ **then**
          $j' =$ `cyclic-next` (j', ccw)
        else
          $break$
return C_{ij}

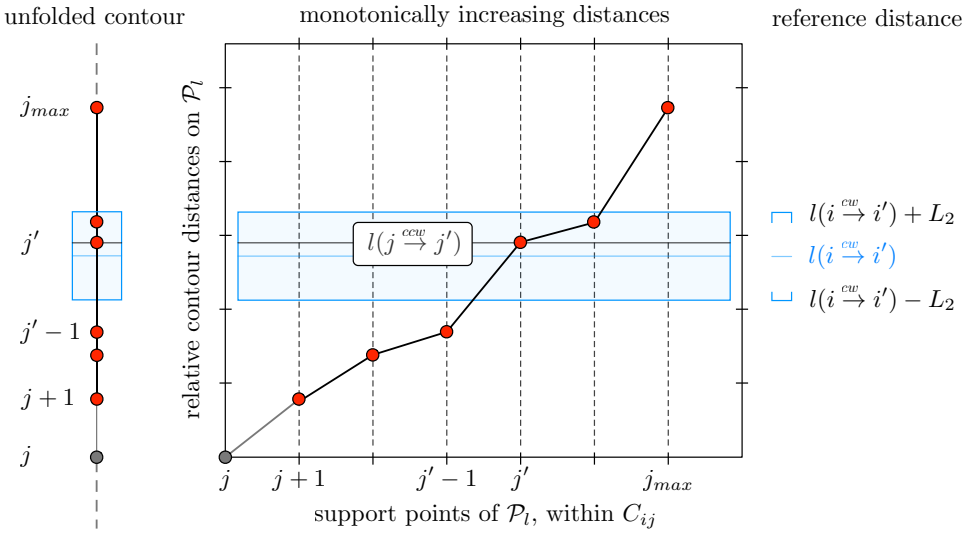


Figure 8.2: Illustration for how candidate sets are constructed. We exploit that contour distances are monotonically increasing functions. Exemplified here is the situation that the contour distance of point $j' - 1$ is too short to make a match with point i' (see line 8 in algorithm `cmp-candidate-set`).

We now discuss algorithm `cmp-candidate-set`, which determines the set of inlier candidates specified in defintion 5. It is based on the fact that, starting from a fixed anchor point, contour distances increase monotonically while traversing a piece's support points. One can take advantage of this observation in order to establish matches between points in the L_1-neighborhoods of pieces. Only if both contour distances are sufficiently similar (up to L_2 pixels), we add a point-pair to the candidate set. Otherwise, if the contour distance on one piece is too short, we move to the current point's immediate successor in the piece's predefined direction. An illustrating example for this situation is given in figure 8.2. Here, the contour distance of point $j' - 1$ is too short for a match with point i'. This case is handled in line 8 of our algorithm. In our illustrating example, moving to the point's successor j' leads to a match in the next iteration.

Some further remarks on the algorithm:

- **Termination:** The while loop in line (7) is executed a finite number of times because, in each iteration, we either move to a point's successor (`cyclic-next`) or else stop (break). Clearly, there is only a finite number of points in either of both L_1-neighborhoods. We denote the number of points in the neighborhoods of i and j by $n_i(L_1) = |N_i(L_1)|$ and $n_j(L_1) = |N_j(L_1)|$, respectively. The algorithm requires at most $n_i + n_j$ iterations, which means that it takes at most twice

as many comparisons of contour distances (in lines (8) and (13)).

- **Early stop:** The algorithm terminates early in lines (12) and (17) if no more candidate can be added to the candidate set. Without loss of generality, let us only consider the first case: Once the if-condition in line (9) is no longer satisfied we know that $j' = j_{max}$, which means that we reached the end of the L_1-neighborhood on $\mathcal{P}_l$. We also know that $l(j \overset{\text{ccw}}{\to} j_{max}) < l(i \overset{\text{cw}}{\to} i') - L_2$. Since $l(i \overset{\text{cw}}{\to} i')$ is a monotonically increasing function in i', it is clear that $l(j \overset{\text{ccw}}{\to} j_{max}) \notin l(i \overset{\text{cw}}{\to} i') \pm L_2$ for any point in clockwise direction from i'. Consequently, no more point can be added to the candidate set.
- **Upper bound:** Furthermore, $n_j(L_1)$ provides an upper bound for the number of candidates in C_{ij}, i.e., $|C_{ij}| \leq n_j$. This is because whenever a new candidate (i', j') is added, we either move to the successor of j' or else terminate if we have already reached j_{max}. Hence any given j' can result in at most one candidate to be added.
- **Practical considerations:** As a closing remark we also want to emphasize that in practice, points are often distributed very heterogeneously along the pieces' observed outer contours. Especially for non-adjacent pieces that share no common adjacent boundary in the original document, one would expect candidate sets to be empty quite often.

8.3.3 Sampling Effectiveness in Comparison with RANSAC

As later discussed in section 8.5, each inlier candidate $(i', j') \in C_{ij}$ allows us to define a rigid transformation. Hence the cardinality of set C_{ij} is the number of hypotheses that need to be considered for the alignment of pieces. We use the quadruple $q = (i, j, i', j')$ to indicate that four points are required to compute such a hypothesis.

To analyze the effectiveness of our algorithm, we compute an estimate for the number of quadruples that need to be examined. Since we do not know whether the anchor points (i, j) form an inlier, we run algorithm `cmp-candidate-set` once for each combination of points. To obtain an estimate, let us make the simplifying assumption that the circumference of the pieces' polygons is c pixels, and let each polygon have n support points. In this case we have n^2 anchor points and the set of quadruples Q is given by:

$$Q = \bigcup_{(i,j)\in\{1,\ldots,n\}^2} \{(i, j, i', j') \mid (i', j') \in C_{ij}\} \tag{8.8}$$

To analyze the average number of quadruples in Q, we first define the neighbor-

hood size $L_1 = \eta c$ proportional to the pieces' circumference with $0 < \eta \ll 1$. In other words, η is the percentage of the pieces' circumference to be considered for the construction of our candidate sets. To obtain a realistic estimate for the cardinality of Q, we need to analyze how many point-correspondences are expected to have similar contour distances. Intuitively, as depicted in figure 8.2, every point i' defines a small sub-range on the L_1-neighborhood of j, and only points within that contour range are eligible for a match with i'. We also know that any given j' can at most result in one point-pair being added to the candidate set. Since the number of inliers in the dataset is negligibly small compared to the number of outliers, support points from different pieces are assumed to be distributed independently of each other. Besides, we also assume that points within both L_1-neighborhoods are distributed uniformly at random and independent of each other. Under these premises, the probability for distance $l(j \overset{\text{ccw}}{\to} j')$ falling into interval $l(i \overset{\text{cw}}{\to} i') \pm L_2$ amounts to $p = 2L_2/L_1$, given $0 < 2L_2 \leq L_1$.

The expected number of points to be matched with the first point $i' = i - 1$ in the L_1-neighborhood of i is thus $n_j(L_1)p$. For the second point $i' = i-2$, only those points j' that were not matched before may contribute to an inlier candidate. For this reason, the expected number of points matched with i' now amounts to $\big[n_j(L_1) - n_j(L_1)p\big]p$. This matching scheme is overall repeated $n_i(L_1)$ many times. The overall number of expected candidates in C_{ij} can be expressed as partial sum of a geometric series:

$$E\big[|C_{ij}|\big] = n_j(L_1)p \sum_{k=0}^{n_i(L_1)-1} (1-p)^k = n_j(L_1)\left[1 - (1-p)^{n_i(L_1)}\right] \tag{8.9}$$

Since each neighborhood describes a fixed proportion η of a piece's circumference through the choice of $L_1 = \eta c$, we can simplify and approximate the above equation by saying that – on average – each such neighborhood will contain ηn many points. If we then take into consideration all n^2 pairs of anchor points we obtain (see proof 1 in the appendix):

$$E\big[|Q|\big] = \eta n^3 p \sum_{k=0}^{\lceil \eta n \rceil - 1} (1-p)^k = \eta n^3 \left[1 - (1-p)^{\lceil \eta n \rceil}\right] \tag{8.10}$$

In practice we empirically chose $\eta = 0.2$ and $L_2 = 10$ pixels. For two pieces with $n = 100$ support points and circumference $c = 1000$ pixels, we have $p = 2L_2/L_1 = 2L_2/(\eta c) = 0.1$ and thus $E\big[|Q|\big] = 0.2 \cdot 100^3 \cdot [1 - 0.9^{20}] \approx 175{,}000$, which is the expected total number of hypotheses generated by running algorithm `cmp-candidate-set`

for all pairs of anchor points. It turned out in our experiments that, despite all of the simplifying assumptions, $E\big[|Q|\big]$ is a very good estimate which only slightly underestimates the true number of initial hypotheses (see section 8.7).

For a fair comparison with RANSAC we determine the number of hypotheses required to obtain a single pair of inlier candidates with a probability of at least $q = 0.95$. We denote by p the chance for obtaining an inlier by random sampling. Assuming that two pieces have approximately 10 inliers and $n = 100$ support points each, the probability would be $p = 0.001$. If we randomly sampled two matching candidates for m rounds, the chance that at least one of the m samples contains two inliers is $1 - (1 - p^2)^m$. Since we require a probability of at least q, we solve $1 - (1 - p^2)^m \geq q$ for m, which gives us:

$$m \geq \frac{\ln(1-q)}{\ln(1-p^2)} = \frac{\ln(0.05)}{\ln(1-10^{-6})} \approx 3 \cdot 10^6 \tag{8.11}$$

The reason why the number of required hypotheses is so high is because the chance of obtaining a pair of inliers is extremely low.

8.4 Distance Transform

So far we have discussed how to determine pairs of inlier candidates between piece-pairs. As pointed out earlier, the resulting set of hypotheses is subject to a number of verification steps in order to invalidate incorrect spatial configurations. In this section we discuss a key concept that forms the basis for this verification.

Measuring distances of points with respect to a polygon is key to the verification of hypotheses. For this reason, we now want to give a short introduction to the *distance transform*, which is a dynamic programming method that allows one to efficiently calculate the spatial separation of points in an image. It computes the distance from each image pixel $\boldsymbol{p} \notin S_k$ to the closest pixel within S_k, where S_k is the set of observed contour points of piece $\mathcal{P}_k$.

Let G define a regular grid (e.g., an image) that is a superset of S_k. Furthermore, let $\mathcal{X}_{S_k}(\boldsymbol{q})$ be the *characteristic function* which yields $+\infty$ if $\boldsymbol{q} \in G \backslash S_k$ and 0 otherwise. The distance transform computes for each given grid point $\boldsymbol{p} \in G$:

$$\mathcal{D}_{S_k}(\boldsymbol{p}) = \min_{\boldsymbol{q} \in G} \big\{ d(\boldsymbol{p}, \boldsymbol{q}) + \mathcal{X}_{S_k}(\boldsymbol{q}) \big\} \tag{8.12}$$

We use the Euclidean distance for $d(\boldsymbol{p}, \boldsymbol{q})$. Note that the above minimization prob-

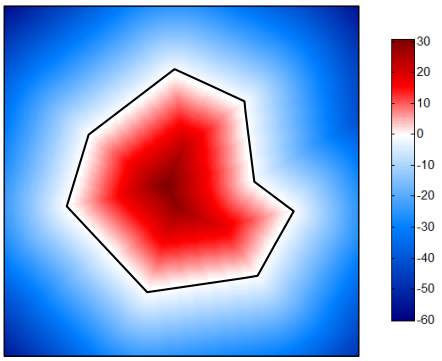


Figure 8.3: Example for a *signed distance map* of a polygon S_k. Each pixel $\boldsymbol{p}$ is assigned its nearest distance to any of the boundary pixels of S_k. According to eq. (8.16), distances for pixels in the foreground region of the polygon ($\boldsymbol{p} \in FG(S_k)$) obtain a positive sign and are color-coded in red; distances for background pixels ($\boldsymbol{p} \in BG(S_k)$) are colored in blue to indicate their negative sign. Note that the polygon is *not* part of the distance map and is only drawn to make distances easier to interpret.

lem can be solved very efficiently: For instance, given an image with n pixels in size, the approach of Felzenszwalb et al. [16] takes only $\mathcal{O}(n)$ time.

We compute a *signed distance map* that distinguishes points on the inside and the outside of a piece. Therefore, let S_k be the observed outer contour of piece $\mathcal{P}_k$, which is here represented by a set of pixel coordinates in the xy-plane. The extrema in x- and y-direction, denoted by $(\underline{x}, \bar{x})$ and $(\underline{y}, \bar{y})$, specify the minimal bounding rectangle of points in S_k. We define the discrete grid G by adding an equal margin of $M = 25$ pixels around each side of the bounding box:

$$G = \left\{ \boldsymbol{p} = (x, y) \in \mathbb{R}^2 \,|\, x \in \{\underline{x} - M, \bar{x} + M\}, y \in \{\underline{y} - M, \bar{y} + M\} \right\} \tag{8.13}$$

Next we determine the subset of pixels lying in the piece's foreground region:

$$FG(S_k) = \left\{ \boldsymbol{p} \in G \,|\, \texttt{inpoly}\,(\boldsymbol{p}, S_k) > 0 \right\} \tag{8.14}$$

Likewise we proceed for background pixels:

$$BG(S_k) = \left\{ \boldsymbol{p} \in G \,|\, \texttt{inpoly}\,(\boldsymbol{p}, S_k) < 0 \right\} \tag{8.15}$$

The procedure `inpoly` returns positive values for pixels on the inside of the polygon, zero for boundary pixels, and negative values on the outside. It can for instance be implemented efficiently using a ray casting algorithm. Due to this partitioning of G into three disjoint sets $FG(S_k)$, $BG(S_k)$ and S_k, we can now compute signed distances

as illustrated in figure 8.3:

$$\mathcal{D}_{S_k}(\boldsymbol{p}) = \left(1_{FG(S_k)}(\boldsymbol{p}) - 1_{BG(S_k)}(\boldsymbol{p})\right) \min_{\boldsymbol{q}\in G} \left\{d(\boldsymbol{p},\boldsymbol{q}) + \mathcal{X}_{S_k}(\boldsymbol{q})\right\} \tag{8.16}$$

Here $1_{S_k} : G \to \{0,1\}$ is the indicator function which takes value 1 for all $\boldsymbol{p} \in S_k$ and 0 otherwise. This yields $\mathcal{D}_{S_k}(\boldsymbol{p}) = 0$ whenever $\boldsymbol{p}$ coincides with a boundary pixel, positive distance values for pixels in the foreground, and negative values for those in the background. Due to its reliance on a discrete grid G, the distance transform is not defined for points $\boldsymbol{p} \notin G$. In this the case we set $\mathcal{D}_{S_k}(\boldsymbol{p}) = -M$, which denotes the minimal distance of any point in the background that is not part of the grid.

8.5 Computing Hypotheses from Candidate Sets

After creating a candidate set once for each pair of anchor points, we next apply a series of rejection and selection steps. Initially we consider the following set of quadruples associated with all anchor points $(i,j) \in I_{\hat{S}_k} \times I_{\hat{S}_l}$:

$$Q = \bigcup_{(i,j)} \left\{(i,j,i',j') \,|\, (i',j') \in C_{ij}\right\} \tag{8.17}$$

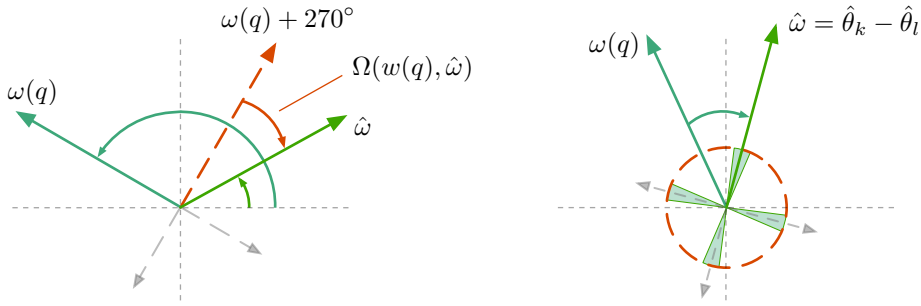
For each $q \in Q$ we then compute a rigid transformation by:

$$H_{k\leftarrow l}(q) = T(\hat{\boldsymbol{s}}_i^k) R(\omega(q)) T(-\hat{\boldsymbol{s}}_i^k) T(\hat{\boldsymbol{s}}_i^k - \hat{\boldsymbol{s}}_j^l) \tag{8.18}$$

We slightly adjust notation here, using the subscript in $H_{k\leftarrow l}(q)$ to denote that the transformation is used for the alignment of piece $\mathcal{P}_l$ with $\mathcal{P}_k$. The reverse alignment direction is represented by $H_{k\to l}(q) = (H_{k\leftarrow l}(q))^{-1}$, which is the inverse of the 3×3 transformation matrix. Note that, unlike for our hill-climbing approach, we now have two inlier candidates. Thus, instead of using the anchor points' immediate successors to eliminate the last degree of freedom, we can now compute rotation angle $\omega(q)$ as the enclosed angle between the two vectors $\hat{\boldsymbol{s}}_{i'}^k - \hat{\boldsymbol{s}}_i^k$ and $\hat{\boldsymbol{s}}_{j'}^l - \hat{\boldsymbol{s}}_j^l$. To fully define a hypothesis, we write $h(q) = (H_{k\leftarrow l}(q), H_{k\to l}(q))$ and use $h = h(q)$ in short when the parameters are irrelevant or clear from the context.

For any piece-pair $(\mathcal{P}_k, \mathcal{P}_l)$ we compute all candidate sets and collect the resulting hypotheses in

$$\mathcal{H}^{(0)} = \bigcup_{(i,j)} \mathcal{H}_{ij}^{(0)}, \tag{8.19}$$



(a) Minimal relative error **(b)** Intervals for acceptance of rotation angle

Figure 8.4: Left: Minimal relative error $\Omega\big(\omega(q), \hat{\omega}\big)$ between two angles. **Right:** Any hypothesis whose rotation angle $\omega(q)$ does not fall into one of four predefined orientation intervals regarding our belief $\hat{\omega}$ is likely to be incorrect. Depending on the confidence of a classifier, these hypotheses are discarded. See text for details.

where $\mathcal{H}_{ij}^{(0)} = \{h(q) \,|\, q = (i, j, i', j'), (i', j') \in C_{ij}\}$ is the subset of hypotheses resulting from the candidate set anchored at (i, j). In the following sections we will describe a sequence of *rejection* and *selection* steps to identify and discard *incorrect* and *redundant* hypotheses:

$$\mathcal{H}^{(0)} \overset{\text{sec. } 8.5.1}{\supseteq} \mathcal{H}^{(1)} \overset{\text{sec. } 8.5.2}{\supseteq} \mathcal{H}^{(2)} \overset{\text{sec. } 8.5.3}{\supseteq} \mathcal{H}^{(3)} \overset{\text{sec. } 8.5.4}{\supseteq} \mathcal{H}^{(4)} \overset{\text{sec. } 8.5.5}{\supseteq} \mathcal{H}^{(5)} \overset{\text{sec. } 8.5.6}{\supseteq} \mathcal{H}^{(6)} \tag{8.20}$$

This chain of steps shrinks the number of hypotheses to only very few rigid transformations that all entail different spatial configurations. Our approach for choosing the most promising configuration from set $\mathcal{H}^{(6)}$ is explained in the next chapter. An overview of how many hypotheses are retained after each stage is given in section 8.7 (see table 8.1).

8.5.1 Conditional Verification for Inconsistent Piece-Orientations

The first filtering operation utilizes content-based information extracted from the foreground regions of two pieces that are to be aligned. It is based on an estimate for the upright direction of each of these pieces. Our approach for computing these estimates is explained in detail in section 8.6. For the moment, let us assume we had already computed two estimates $\hat{\theta}_k$ and $\hat{\theta}_l$, one for each piece $\mathcal{P}_k$ and $\mathcal{P}_l$.

Relying on these estimates, our belief about what is likely to be the correct rotation angle is computed as $\hat{\omega} = \hat{\theta}_k - \hat{\theta}_l$, which denotes the smallest sign-preserving enclosed angle between the pieces' orientation estimates. For any given hypothesis computed from quadruple q we verify whether the hypothesis's rotation $\omega(q)$ is consistent with our belief $\hat{\omega}$.

Due to our estimation procedure, the obtained orientation can be classified into incorrect, correct, or correct up to $90°$. We note that without a costly semantic analysis, text lines only allow us to reliably estimate an orientation, but *not* the direction. Other examples that cause similar ambiguities are straight lines and content elements such as tables, for which there is often no telling whether they are directed horizontally or vertically. To deal with this kind of uncertainty, we define the *minimal relative error*, which measures the difference between two angles, up to $90°$:

$$\Omega\big(\omega(q), \hat{\omega}\big) = \min_{k=0,\dots,3} \Big\{ \hat{\omega} - \big(\omega(q) + k \cdot 90°\big) \Big\} \in [0°, 45°] \tag{8.21}$$

The above equation computes the difference between two angles as the smallest sign-preserving enclosed angle, which is illustrated in figure 8.4a. At this point, using the error of $\omega(q)$ with regards to our belief $\hat{\omega}$, one could already invalidate incorrect hypotheses. However, since correct orientation estimates can not be obtained for every piece-pair, this strategy also potentially rejects correct hypotheses. For example, this is the case whenever pieces are colored mostly uniformly (without any content), thus providing no meaningful orientation estimates. To handle this matter properly, we propose to learn a linear classifier (see section 8.6) to decide on the reliability of the pieces' orientation estimates. Let $c(\hat{\theta}_k), c(\hat{\theta}_l)$ be the classifier's confidence for the reliability of estimates $\hat{\theta}_k$ and $\hat{\theta}_l$. For each hypothesis $h(q)$ obtained from the candidate set we then evaluate:

$$\Omega\big(\omega(q), \hat{\omega}\big) \leq \eta_\omega \vee \neg\big[c(\hat{\theta}_k) > T \wedge c(\hat{\theta}_l) > T\big] \tag{8.22}$$

The hypothesis is retained if this boolean expression evaluates to *true* and is discarded otherwise. That is, we *reject* $h(q)$ if and only if the minimal relative error is too large *and* the classifier's confidence is above threshold T for both pieces. In practice we use a threshold $\eta_\omega = 1°$ on the error. Given that $\hat{\omega}$ is correct, this rule only invalidates hypotheses which are most likely incorrect, because an incorrect rotation angle inevitably leads to an incorrect spatial configuration of pieces. We illustrate this idea in figure 8.4b. Assuming that $c(\hat{\theta}_k) > T \wedge c(\hat{\theta}) > T$ evaluates to *true*, the hypothesis can be discarded because its rotation angle $w(q)$ does not fall into one of the four orientation intervals regarding $\hat{\omega}$ (green wedges). In summary, hypotheses retained after this first rejection step are given by:

$$\mathcal{H}_{ij}^{(1)} = \Big\{ h(q) \in \mathcal{H}_{ij}^{(0)} \,|\, \Omega\big(\omega(q), \hat{\omega}\big) \leq \eta_\omega \vee \neg\big[c(\hat{\theta}_k) > T \wedge c(\hat{\theta}_l) > T\big] \Big\} \tag{8.23}$$

Finally, the hypotheses from all candidate sets are collected in $\mathcal{H}^{(1)} = \bigcup_{(i,j)} \mathcal{H}_{ij}^{(1)}$.

8.5.2 Local Verification for Overlap

Our second rejection step is based on the fact that well aligned pieces must not overlap each other. We test this by verifying that none of the support points within candidate set C_{ij} is projected into the foreground region of the respective other piece. To determine whether this is the case, we use the signed distance of each point in C_{ij} to the nearest boundary point of the other piece. That is, using the pieces' signed distance maps introduced in section 8.4, we check if either

$$\max_{(i',j')\in C_{ij}} \left\{ \mathcal{D}_{S_k}(H_{k\leftarrow l}\hat{\boldsymbol{s}}^l_{j'}) \right\} > \eta_{FG} \tag{8.24}$$

or

$$\max_{(i',j')\in C_{ij}} \left\{ \mathcal{D}_{S_l}(H_{k\rightarrow l}\hat{\boldsymbol{s}}^k_{i'}) \right\} > \eta_{FG} \tag{8.25}$$

holds. That is, if a single point from the candidate set exceeds the threshold, hypothesis $h \in \mathcal{H}^{(1)}_{ij}$ is rejected. We empirically set the threshold to $\eta_{FG} = 10$ pixels. By $H_{k\leftarrow l}\hat{\boldsymbol{s}}^l_j$ we refer to the updated coordinates of the j-th support point of piece $\mathcal{P}_l$. Note that for piece $\mathcal{P}_k$ one has to use the inverse transformation instead.

This verification is extremely fast: Aside from computations of point coordinates, checks in eq. (8.24) and (8.25) only require lookups in the pieces' distance maps, which can be computed once in advance. Note that this rejection step only checks for *local* overlaps along short contour segments. All locally verified hypotheses are finally collected in $\mathcal{H}^{(2)} = \bigcup_{(i,j)} \mathcal{H}^{(2)}_{ij}$.

8.5.3 Resolving Local Ambiguity

Since the anchor points are fixed to (i, j) for all hypotheses from the same candidate set, all of its entailed spatial configurations can vary only in the orientation of piece $\mathcal{P}_l$. Arguably, most of these hypotheses are very likely redundant. We thus choose the best representative from each candidate set and discard the others.

Along the lines of the discussion in section 8.2, we use a truncated squared Euclidean distance to disambiguate inliers and assign a fixed penalty to outliers:

$$\Delta_L\big(h; C_{ij}\big) = \sum_{(i',j')\in C_{ij}} \Big\lfloor \|\hat{\boldsymbol{s}}^k_{i'} - H_{k\leftarrow l}\hat{\boldsymbol{s}}^l_{j'}\|^2, \eta_{out} \Big\rfloor \tag{8.26}$$

Recall from section 3.4 that $\lfloor \cdot, \cdot \rfloor$ denotes the minimum of two real values. If points of an inlier candidate have a squared distance of $\eta_{out} = 5$ or more, they are considered as an outlier. Note that this local error function is not suitable for comparing hypothe-

ses among different candidate sets because the number of their correspondences tends to vary considerably.

According to eq. (8.26) we then choose the most accurate hypothesis, individually per candidate set. Considering each pair of anchor points $(i, j) \in I_{\hat{S}_k} \times I_{\hat{S}_l}$ gives:

$$\mathcal{H}^{(3)} = \bigcup_{(i,j)} \left\{ \operatorname*{argmin}_{h \in \mathcal{H}_{ij}^{(2)}} \{ \Delta_L(h; C_{ij}) \} \right\} \tag{8.27}$$

Since each candidate set now contributes at most one hypothesis, the cardinality of set $\mathcal{H}^{(3)}$ is upper bounded by $n(k) \times n(l)$, i.e., the product of the number of support points on pieces $\mathcal{P}_k$ and $\mathcal{P}_l$.

8.5.4 Global Verification for Overlap

Since hypotheses in $\mathcal{H}^{(3)}$ stem from different candidate sets, they typically feature a large variety of spatial configurations. Also recall that so far, each hypothesis has been verified only locally in terms of content overlap. Thus, the current set of hypotheses may still contain transformations that entail incorrect spatial configurations, which can only be invalidated when considering both pieces as a whole. To remove these inconsistent hypotheses, we re-use the overlap constraints from eq. (8.24) and (8.25). We now test all remaining support points that have not been checked for content overlap yet, i.e., all and only those points that are not part of the candidate set. Any hypothesis passing this global verification puts pieces into a non-overlapping configuration and is kept for later use in $\mathcal{H}^{(4)}$.

8.5.5 Verification for Gaps along Adjacent Boundaries

For our next rejection step we first need to determine the pieces' adjacent boundary regions after the alignment. Identifying those regions allows us to test whether there are gaps in between the two pieces. We found gaps to be a reliable indicator for incorrect hypotheses and, consequently, such hypotheses should be rejected.

Identifying Adjacent Boundary Regions

In the following we demonstrate how the adjacent boundary region can be determined for piece $\mathcal{P}_k$. Given a hypothesis $h \in \mathcal{H}^{(4)}$, the first step is to identify the subset of points $A_k(h)$ that become adjacent to piece $\mathcal{P}_l$ through h. We can accomplish this by

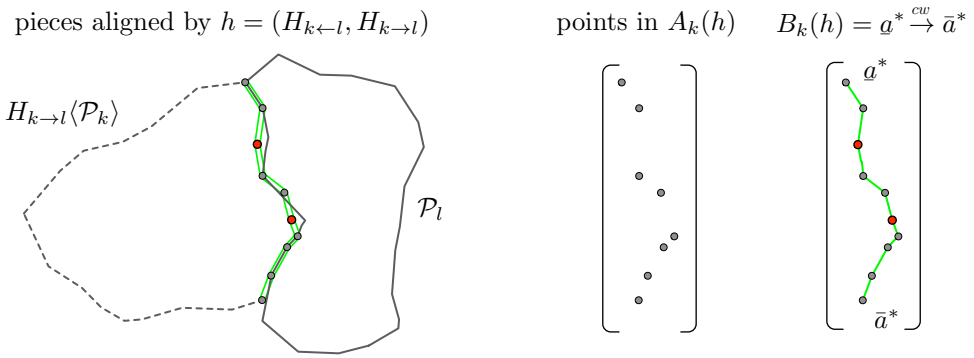


Figure 8.5: Illustration of an adjacent boundary region (green segment) on piece $\mathcal{P}_k$, which is entailed by hypothesis h. The two non-adjacent support points (red) indicate a gap between the pieces. The polygonal chain $B_k(h)$ on the righthand side represents the adjacent boundary region on piece $\mathcal{P}_k$ after alignment with $\mathcal{P}_l$.

using our precomputed distance maps:

$$A_k(h) = \left\{ i \in \{1, \ldots, n(k)\} \mid -\eta_{BG} \leq \mathcal{D}_{S_l}(H_{k\rightarrow l}\hat{\boldsymbol{s}}_i^k) \leq \eta_{FG} \right\} \tag{8.28}$$

We empirically chose the parameter $\eta_{BG} = \frac{1}{2}\eta_{FG} = 5$ based on the threshold for content overlap but made it a bit more conservative to identify even small gaps. As illustrated in figure 8.5, all points in $A_k(h)$ are in close vicinity to the contour of piece $\mathcal{P}_l$. The two red circles correspond to support points which are too distant from the contour, indicating a gap between the two pieces.

Note that A_k is a subset of $I_{\hat{S}_k}$ and thus itself is a cyclically ordered set. Because of that, we do not know the delimiting points of the polygonal chain that defines the adjacent boundary region (see definition 2). We solve this problem by once considering each possible choice for start point $\underline{a} \in A_k(h)$ and end point $\bar{a} \in A_k(h)$: We first pick a start point and then choose the end point as $\underline{a}$'s clockwise predecessor in the cycle $A_k(h)$. This gives us $|A_k(h)|$ many polygonal chains $\underline{a} \overset{\text{cw}}{\rightarrow} \bar{a}$. If the points are chosen incorrectly, we will encounter some point $i \in \underline{a} \overset{\text{cw}}{\rightarrow} \bar{a}$ while traversing the contour from $\underline{a}$ to $\bar{a}$ which is *not* spatially close to the contour of $\mathcal{P}_l$. The correct polygonal chain is the one whose points are all in close proximity to the other piece:

$$(\underline{a}^*, \bar{a}^*) = \underset{\underline{a},\bar{a}}{\operatorname{argmin}} \left\{ \max_{i \in \underline{a} \overset{\text{cw}}{\rightarrow} \bar{a}} \{ -\lfloor \mathcal{D}_{S_l}(H_{k\rightarrow l}\hat{\boldsymbol{s}}_i^k), 0 \rfloor \} \right\} \tag{8.29}$$

The chain defined by $(\underline{a}^*, \bar{a}^*)$ minimizes the maximal absolute distance of any support point i on the chain with respect to the other piece's background. Thus, testing

all combinations gives us one polygonal chain $B_k(h) = \underline{a}^* \overset{\text{cw}}{\rightarrow} \bar{a}^*$ which is positioned in close vicinity to the other piece (see figure 8.5). Our choice in eq. (8.29) is optimal because for any other two points we would traverse parts of the boundary of piece $\mathcal{P}_k$ that are non-adjacent regarding $\mathcal{P}_l$. In the following we call $B_k(h)$ the *adjacent boundary region* on piece $\mathcal{P}_k$ entailed by hypothesis h.

Verification for Gaps

Having identified the adjacent boundary region makes the verification for gaps very straightforward: Whenever $B_k(h) \supset A_k(h)$ holds, we know that some support points are more distant than η_{BG} pixels from the other piece's contour (in the background). In this case our polygonal chain contains non-adjacent points that constitute a *gap* in between the two pieces. Finally, we reject any hypothesis $h \in \mathcal{H}^{(4)}$ that entails a gap larger than $\eta_{gap} = 5$ pixels:

$$\max_{i \in B_k(h)} \{-\mathcal{D}_{S_l}(H_{k \to l} \hat{\boldsymbol{s}}_i^k)\} > \eta_{gap} \tag{8.30}$$

Note that the verification for gaps is performed separately *for both pieces* and only hypotheses satisfying eq. (8.30) in both cases are added to $\mathcal{H}^{(5)}$.

8.5.6 Symmetric Embedding Error

For our last selection step we introduce an error function $\Delta_G(h)$ to estimate how accurately two pieces are aligned though hypothesis $h \in \mathcal{H}^{(5)}$. The key idea is that the quality of a hypothesis depends on how many support points from either piece are projected to the close vicinity of the other piece's outer contour. In compliance with the strategy employed in MSAC we assign scores to support points according to their proximity. To this end, we now use pieces as a whole, considering *all* support points for the computation of a *symmetric embedding error*:

$$\begin{aligned} \Delta_G(h) = &\sum_{j=1}^{n(l)} \delta\big(\mathcal{D}_{S_k}(H_{k \leftarrow l} \hat{\boldsymbol{s}}_j^l)\big) \\ &+ \sum_{i=1}^{n(k)} \delta\big(\mathcal{D}_{S_l}(H_{k \to l} \hat{\boldsymbol{s}}_i^k)\big) \end{aligned} \tag{8.31}$$

This function quantifies the error for embedding two closed polygonal curves (i.e., the contour of the two associated pieces) into a joint coordinate system. The function is symmetric in that it evaluates the "adjacency" of one aligned piece given the other and

vice versa. The cost function δ in eq. (8.31) assigns low penalties to all well-embedded support points (i.e., those positioned in close proximity to the respective other piece's polygon), and every "outlier" (i.e, a non-adjacent point) exceeding a given threshold regarding the signed nearest distance scores a constant penalty:

$$\delta(d) = \left\lceil \lfloor d, D_{FG} \rfloor^{e_{FG}}, 0 \right\rceil + \left\lceil \lfloor -d, D_{BG} \rfloor^{e_{BG}}, 0 \right\rceil \tag{8.32}$$

As discussed in section 8.4, the sign of distance d distinguishes points on the inside of a polygon (content overlap) from points on the outside (gap). Note that exactly one of the above two terms yields a non-negative value while the other one is 0. The reason is that any point can either be located in the foreground region ($d > 0$), the background region ($d < 0$), or else be positioned exactly on the contour of the other piece. In the latter case we obtain signed a distance $d = 0$ and thus eq. (8.32) yields $\delta(d) = 0$.

The parameters D_{FG} and D_{BG} control the truncation distance for embedded points on the inside and outside of the polygon, respectively. Exponents e_{FG} and e_{BG} define the surface of cost function δ, and choosing different values provides an asymmetric cost function. Based on preliminary experiments we empirically chose $D_{FG} = D_{BG} = 5$ pixels as the truncation distance. Exponents are set to $e_{FG} = 2$ and $e_{BG} = 1$ to penalize content overlap slightly stronger than spatially disconnected boundaries, as this may often occur for pieces suffering from material loss. Also note that the symmetric embedding error is suitable for comparing hypotheses from different candidate sets because it takes into account all support points from both pieces.

8.5.7 Non-Maximum Suppression

In practice it is beneficial to not only determine a single hypothesis, but rather a small diverse set resembling different spatial configurations. We accomplish this by using algorithm `nms-top-K`. It computes the top K non-maximum suppressed hypotheses with respect to the symmetric embedding error. We want to point out that if $K = 1$, only a single hypothesis $\hat{h} = \operatorname{argmax}_{h \in \mathcal{H}^{(5)}} \{\Delta_G(h)\}$ is added to the final set $\mathcal{H}^{(6)}$. If on the other hand $K > 1$, we add multiple hypotheses.

An example for the top 5 non-maximum suppressed hypotheses is presented in figure 8.6. To decide whether two hypotheses are sufficiently different from each other, algorithm `nms-top-K` makes use of the the overlap between adjacent boundary regions entailed by two hypotheses. This criterion will be discussed next.

Figure 8.6: Example for a piece-pair with its top 5 non-maximum suppressed hypotheses, arranged from left to right in ascending order regarding their symmetric embedding error (see eq. (8.31)). The best hypothesis is highlighted in green.

8.5.8 Min-Overlap Entailed by Pairs of Hypotheses

We quantify the similarity of two spatial configurations, resulting from two hypotheses h and h', indirectly by means of the overlap between their entailed adjacent boundary regions. The idea is that two pieces become arranged similarly by two hypotheses if and only if the adjacent boundary regions entailed by h, on each piece, are similar to those entailed by h'. Accordingly, we define the hypotheses' *min-overlap* by

$$\Gamma_{k,l}(h,h') = \left\lfloor \gamma\big(B_k(h), B_k(h')\big), \gamma\big(B_l(h), B_l(h')\big) \right\rfloor, \tag{8.33}$$

where

$$\gamma(B,B') = \frac{l(B \cap B')}{l(B) + l(B') - l(B \cap B')} \in [0,1]. \tag{8.34}$$

The above definition is analogous to how overlap criteria are used in many other computer vision applications. It is common to compute the overlap as the intersection over the union because it equally penalizes hypotheses that under- or over-estimate the ground truth. The key distinction to other applications is that we have to consider two "predictions" entailed by each hypothesis, i.e., one adjacent boundary region on each piece. Thus, we compute the *intersection over union* once separately for each of the two pieces: In eq. (8.34), the numerator is the length of the polygonal chain resulting from intersecting the two chains that represent the pieces' adjacent boundary regions. In the same manner we proceed for the denominator, for which the intersecting part of the two chains needs to be subtracted because otherwise it would be counted twice. Obviously, if $B = B'$ holds, we obtain $\gamma(B, B') = 1$ as intended.

If two h and h' entail similar spatial configurations, the min-overlap is close to 1. If on the other hand the overlap is high for one piece but 0 for the other, the min-overlap correctly reflects this by returning 0.

Algorithm 8.2: Non-maxmimum suppressed hypotheses (`nms-top-K`)

Input : Pieces $\mathcal{P}_k$ and $\mathcal{P}_l$, hypotheses $\mathcal{H}^{(5)}$
Parameters : Desired number of hypotheses K
Output : Top K non-maximum suppressed hypotheses $\mathcal{H}^{(6)}$

Initialization
  $\mathcal{H}^{(6)} \leftarrow \emptyset$

Non-Maximum Suppression
  while $|\mathcal{H}^{(6)}| < K$ **do**
    $\mathcal{H} = \mathcal{H}^{(5)} \setminus \mathcal{H}^{(6)}$
    `/* no more hypotheses remain */`
    if $|\mathcal{H}| = 0$ **then**
      break
    `/* choose minimizer from remaining hypotheses */`
    $\hat{h} = \text{argmin}_{h \in \mathcal{H}}\{\Delta_G(h)\}$
    `/* verify that spatial configuration is different */`
    $nms = false$
    foreach $h \in \mathcal{H}^{(6)}$ **do**
      `/* min-overlap between entailed adjacent boundary regions is too large */`
      if $\Gamma_{k,l}(\hat{h}, h) > 0.2$ **then**
        $nms = true$
        break
    if nms **then**
      `/* ignore hypothesis (near-duplicate) */`
      $\mathcal{H}^{(5)} = \mathcal{H}^{(5)} \setminus \{\hat{h}\}$
    else
      `/* add to set of hypotheses */`
      $\mathcal{H}^{(6)} = \mathcal{H}^{(6)} \cup \{\hat{h}\}$
return $\mathcal{H}^{(6)}$

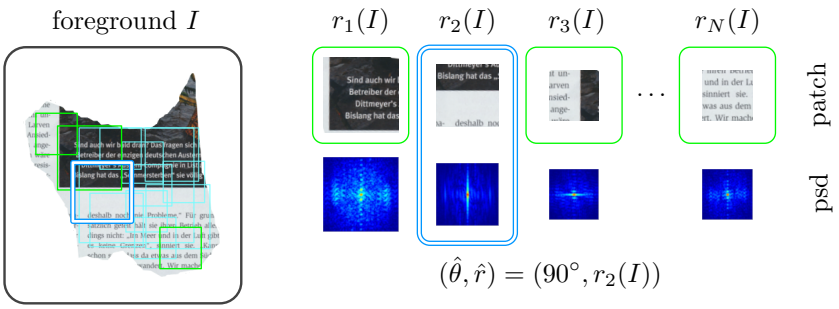


Figure 8.7: Left: Patches of different sizes are positioned greedily to cover foreground region I. **Right:** Below a few highlighted patches we show their power spectral density (psd). Estimate $\hat{\theta}$ for the piece's orientation always stems from a single patch $\hat{r}$, which is chosen according to eq. (8.36).

8.6 Conditionally used Orientation Estimates

This section discusses the necessary steps to obtain an orientation estimate from the content of each piece, which we have previously utilized in the first rejection step (see section 8.5.1). We commence with an explanation of how these orientation estimates can be obtained from the Fourier transform of image patches. After that, we explain how supervised learning can be used to invalidate incorrect estimates.

8.6.1 Orientation Estimates from the Fourier Transform

We briefly recap the technique of Hollitt and Deeb [29] to estimate the dominant orientation of an image. The idea is to find the direction along which the image shows the greatest variation in intensity values. Usually, its *upright direction* is either identical or orthogonal to that direction. Instead of working in the spatial domain, the authors apply a Fourier transform on the image to find its dominant orientation in the frequency domain. Therefore, let $I(x, y) \in \mathbb{N}^{n \times n}$ denote an image, and let $(\mathcal{F}I)(\xi_x, \xi_y) \in \mathbb{C}^{n \times n}$ be its Fourier transform. Instead of parametrizing in terms of ξ_x, ξ_y, which are conjugate to axes x and y, one can equivalently consider the Fourier transform to be a function of polar coordinates ξ_ρ, ξ_θ. To find the direction of strongest spatial variation, we sum over the *power spectral density (psd)* along θ:

$$g(\theta; I) = \sum_{\xi_\rho} |(\mathcal{F}I)_m(\xi_\rho, \xi_\theta)|^2 \tag{8.35}$$

The term $(\mathcal{F}I)_m$ refers to the psd after applying a band-pass filter, which combines a low-pass with a high-pass filter. The low-pass filter avoids a bias for larger values at

orientations near $45°$ of the image axes, and the high-pass filter eliminates illumination changes.

Using a piece's entire image would most likely lead to inaccurate or even entirely incorrect results, because strong gradients at its outer contour are uninformative for the true upright direction. To account for the fact that we are dealing with arbitrarily shaped pieces, we have to concentrate on their foreground region. However, some of these regions may be detrimental for the computation, e.g., if parts of them cover a natural image. Since we do not know the optimal foreground sub-region beforehand, we first use a sliding window approach to position patches of varying size. As shown in figure 8.7, each patch covers a square region of interest on foreground region of image I. We greedily position patches $\left[r_i(I)\right]_{i=1\ldots N}$ one by one, from large to small. A new patch is placed only if its mutual overlap with any of the previously positioned patches is not larger than 20%. Finally, we compute the Fourier transform, separately for each patch, and choose the orientation that shows the strongest spatial variation among all orientations and patches:

$$(\hat{\theta}, \hat{r}) = \underset{\theta, r_i(I)}{\operatorname{argmax}} \left\{ g(\theta; r_i(I)) / B_i \right\} \tag{8.36}$$

The bandwidth B_i is computed as the difference between the upper and the lower cutoff frequency, and both are chosen with respect to the patch size. Normalizing by B_i is necessary to avoid a systematic bias for larger patches.

8.6.2 Discriminative Model for Orientation Estimates

Since eq. (8.36) always yields an orientation estimate, we now turn to learn a discriminative model (SVM) to decide whether an estimate should be trusted or needs to be invalidated.

We found that a strong peak $g(\hat{\theta}; \hat{r})$ indicates a robust estimate and hence estimate $\hat{\theta}$ should not be discarded. In contrast, we need to reject ambiguous orientation estimates in cases where multiple peaks occur (e.g., for textured image regions). We formalize this idea by sampling sums of spectral densities along different directions $\theta \in [\hat{\theta} - 45°, \hat{\theta} + 45°]$ in steps of $\Delta\omega = 0.5°$, using eq. (8.35). This approach leads to a 181-dimensional descriptor:

$$\boldsymbol{\vartheta}_{ori}[90 \pm i] = g(\hat{\theta} \pm i \cdot \Delta\omega; \hat{r}) \,/\, g(\hat{\theta}; \hat{r}), \;\; \forall i \in \{0, \ldots, 90\} \tag{8.37}$$

By centering the descriptor around $\hat{\theta}$ we make it comparable with descriptors of

other patches because they all have their peak at the same position. Furthermore, we normalize it by $g(\hat{\theta}; \hat{r})$ to obtain a characterization of the relative strength of the peak. To gain partial invariance against angle shifts we further aggregate descriptor values into blocks of $5°$, $15°$, and $45°$, respectively. For each of these $2 \times (9 + 3 + 1)$ blocks we compute its mean and standard deviation and append these values to $\boldsymbol{\vartheta}_{ori}$. Note that all the information needed is readily available from the spectral densities computed in eq. (8.36). Finally, to complement this representation from the frequency domain, we extract Haralick texture features [26][1] on patch $\hat{r}$ and add them to our final descriptor of $(181 + 26 + 4)$ values.

To distinguish correct from incorrect orientation estimates, we then train a linear support vector machine using SVMLight [31]. For this purpose, we categorize patches into positive and negative training examples. As first step we compute orientations $\hat{\theta}$ of randomly oriented pieces according to eq. (8.36) while keeping track of the error regarding their ideal upright directions θ^* that are known from the ground truth. Since our approach works in the frequency domain, $\hat{\theta}$ often fails to give a piece's upright *direction*; however, it may still identify its correct *orientation*. That is, in some cases, text or straight lines can cause the estimate to be correct modulo $90°$. Because of that, we categorize training examples into positive and negative examples based on their minimal relative error, i.e., we compute $\Omega(\hat{\theta}, \theta^*) \in [0°, 45°]$.

Recall from eq. (8.21) that the minimal relative error is zero if and only if the orientation estimate $\hat{\theta}$ is correct modulo $90°$. We found that our orientation estimates are very precise for the majority of examples. If $\Omega(\hat{\theta}, \theta^*)$ is less than $2°$, we consider the descriptor of the associated patch $\hat{r}$ to be a positive example – otherwise it is used as a negative example.

8.6.3 Parameter Tuning

Next we tune the performance of our linear classifier by adjusting the decision threshold of the SVM. For this purpose, we perform cross-validation on the basis of piece-pairs. Recall that the purpose of the rejection step in section 8.5.1 was to exclusively reject incorrect hypotheses based on our orientation estimates. To ensure that this is the case, two conditions must hold: (i) Both orientation estimates $\hat{\theta}_k$ and $\hat{\theta}_l$ must be correct up to $90°$ in the first place, and (ii) the classifier's confidence regarding the correctness of these estimates must be sufficiently high. In that section we formalized this second aspect in eq. (8.22) by $c(\hat{\theta}_k) > T \wedge c(\hat{\theta}_l) > T$. By c we denoted the classifier's

[1] We use *angular second moment*, *contrast*, *correlation*, and *entropy*.

confidence in the reliability of the estimate and T is the decision threshold that is to be tuned to achieve optimal results.

Depending on T and the correctness of the orientation estimates, we distinguish two outcomes for any given piece-pair. Two pieces with correct estimates are called a *joint positive* example. For these pairwise examples we want the SVM to give a positive prediction for *both* pieces. Accordingly, if both confidences $c(\hat{\theta}_k)$ and $c(\hat{\theta}_l)$ exceed threshold T, we have a *joint true positive* as the outcome. On the other hand, if any of the two pieces is assigned an incorrect estimate, but the SVM still classifies both as positive, we call the outcome a *joint false positive*. Tuning the threshold aims to avoid joint false positives because these inevitably lead to a rejection of *correct* hypotheses.

We perform cross-validation on piece-pairs from $\{\text{train}\} + \{\text{val}\}$ of the *bdw082010* dataset to optimize our classifier's performance. If we would set $T = -\infty$, all orientation estimates would be predicted to be correct. In this case we achieve a recall of 1.0, however, the precision is only 0.7230. That is, in 72.30% of the cases for which we decided to trust the estimates, this decision would actually be correct. Correspondingly, for the other 27.70% of our pairwise examples, we would falsely classify incorrect estimates as being correct, leading to incorrect hypotheses. The other extreme case would be to set $T = +\infty$, which corresponds to the special case of accepting all hypotheses. Based on the cross-validation results we empirically chose $T = 0.4$ to account for the fact that precision is more important than recall. For this choice of T we achieve a precision of 0.8864 at a recall of 0.8469.

Recall from section 8.5.1 that, based on the pieces' orientation estimates, hypotheses can only be rejected if condition $c(\hat{\theta}_k) > 0.4 \wedge c(\hat{\theta}_l) > 0.4$ is satisfied, i.e., only if the classification outcome is either a joint true positive or a joint false positive (if the classifier's confidence is too low, the hypothesis is always kept). Overall, this requirement is met for 69.08% of our cross-validation examples. In this scenario, hypotheses are accepted if their minimal relative error is sufficiently small and discarded otherwise. This becomes evident when considering eq. (8.22) again:

$$\Omega\big(\omega(q), \hat{\omega}\big) \leq \eta_\omega \vee \neg\big[c(\hat{\theta}_k) > T \wedge c(\hat{\theta}_l) > T\big] \tag{8.38}$$

$$\Leftrightarrow \Omega\big(\omega(q), \hat{\omega}\big) \leq \eta_\omega \vee \neg\big[true\big] \tag{8.39}$$

$$\Leftrightarrow \Omega\big(\omega(q), \hat{\omega}\big) \leq \eta_\omega \tag{8.40}$$

After choosing the threshold, we re-train the model on the full $\{\text{train}\}+\{\text{val}\}$ datasplit. We use this final model with $T = 0.4$ for all of the following experiments.

8.7 Evaluation

In this section we focus on the evaluation of our proposed MSAC variant. Our new evaluation scheme is conceptually very similar to the one used for hill-climbing (see chapter 6). The key distinction is that the adjustment cost is replaced by a min-overlap criterion. While also being based on the ground truth, our new performance measure is much easier to comprehend and thus more straightforward to use. After discussing the evaluation methodology briefly, we conduct several experiments to draw a direct comparison between hill-climbing and MSAC in light of the new min-overlap based performance measure.

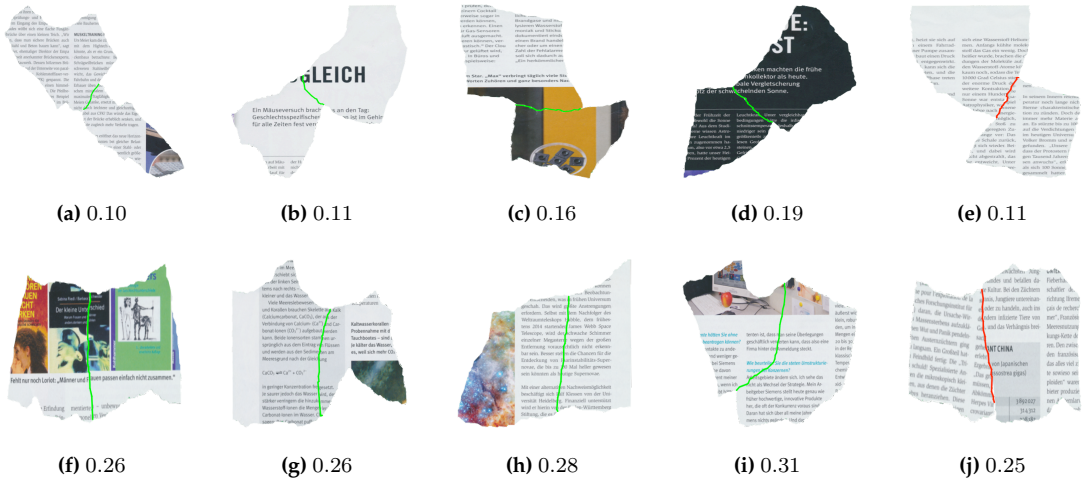
8.7.1 Methodology

Our new evaluation measures the degree of overlap of the pieces' adjacent boundary regions with the annotation given in the ground truth. To evaluate the quality of the hypotheses obtained from our MSAC method, the prevailing question is: How does the *min-overlap* change as a function of *recall* (i.e., the percentage of piece-pairs). We want to emphasize that this evaluation is very similar to the one based on adjustment cost (see section 6.4): A high min-overlap occurs if two pieces are adjacent along the same boundary regions as specified in the ground truth. Therefore, a high min-overlap implies low adjustment cost, and vice versa.

8.7.2 Computing Min-Overlap based on Ground Truth

Recall from section 8.5.8 that the min-overlap for two pieces is close to 1 if two transformations h and h' entail very similar spatial configurations. On the other hand, it is 0 once that the adjacent contour regions on either piece are non-overlapping. Two illustrating examples for adjacent piece-pairs that were not aligned correctly are shown in the right column of figure 8.8. In both cases, the min-overlap is 0 because the adjacent boundary regions (red polygonal curves) are completely different to the annotation in the ground truth.

So far we have already seen one application of min-overlap: In the context of non-maximum suppression, the criterion was used to ensure that *two predicted* hypotheses h and h' entail sufficiently different spatial configurations. For evaluation purposes we now consider only *one predicted* hypothesis h and determine the second hypothesis h^* from the ground truth. Recall from section 3.8 that G_k and G_l are transformations that position pieces as in the manually reconstructed document page. Accordingly, $G_k\langle\mathcal{P}_k\rangle$



(a) 0.10 **(b)** 0.11 **(c)** 0.16 **(d)** 0.19 **(e)** 0.11

(f) 0.26 **(g)** 0.26 **(h)** 0.28 **(i)** 0.31 **(j)** 0.25

Figure 8.8: Examples for piece-pairs with varying max-connectivity, written underneath each example. A green polyline is used to indicate that the min-overlap is 0.5 or higher. Examples in the top row a-d and bottom row f-i depict piece-pairs with low–medium and medium–high connectivity, respectively. Contrary to these examples with a min-overlap close to 1, the two examples in e and j have min-overlap 0, which reflects that they were not aligned correctly.

and $G_l\langle\mathcal{P}_l\rangle$ are the pieces' aligned versions as specified in the ground truth. We can thus compute a single transformation for aligning $\mathcal{P}_l$ with $\mathcal{P}_k$ by $G_{k\leftarrow l} = G_k^{-1}G_l$. Using $h^* = (G_{k\to l}, G_{k\leftarrow l})$ in eq. (8.33) then gives the *min-overlap* of any predicted hypothesis $\hat{h}$ with the ground truth[1], which we denote by $\Gamma_{k,l}(\hat{h}, h^*)$.

Because $B_k(h^*) = B_l(h^*) = \emptyset$ holds for non-adjacent pieces, it now becomes apparent that the min-overlap always equals 0 in this case. This is consistent with our conception that non-adjacent pieces can not be aligned correctly. On the other hand, if the two pieces were adjacent, the min-overlap takes values from $[0, 1]$ depending on the predicted hypothesis, and a higher value indicates a more accurate alignment.

8.7.3 Computing Max-Connectivity based on Ground Truth

For our evaluation we introduce a second concept called *max-connectivity*, which measures the "degree of adjacency" between pieces in the ground truth. To compute the max-connectivity of two pieces, we compute the length of the adjacent boundary region on either piece (numerator) relative to its circumference (denominator). The *max-*

[1] In practice, the adjacent boundary regions are precomputed based on repositioned pieces $G_k\langle\mathcal{P}_k\rangle$ and $G_l\langle\mathcal{P}_l\rangle$ and also become stored in the ground truth.

connectivity is then obtained as the maximum of the two resulting ratios:

$$\Lambda_{k,l}(h^*) = \left\lceil \frac{l(B_k(h^*))}{l(\hat{S}_k)}, \frac{l(B_l(h^*))}{l(\hat{S}_l)} \right\rceil \in [0,1] \tag{8.41}$$

In analogy with the contour distance of polygonal chains defined in eq. (8.3), $l(\hat{S}_k)$ and $l(\hat{S}_l)$ denote the lengths of the pieces' closed polygonal curves. Note that, in contrast to the min-overlap, we here use the max instead of the min operator. The reasoning is that, if a small piece is adjacent to a large piece, we want their max-connectivity to be determined with respect to the small piece's circumference. Otherwise, all adjacency relationships involving small pieces would be biased towards low connectivity values.

We give a few examples for piece-pairs with varying connectivity in figure 8.8. As can be seen from those examples, correctly aligning pieces with low connectivity is inherently more difficult in comparison to strongly adjacent pieces.

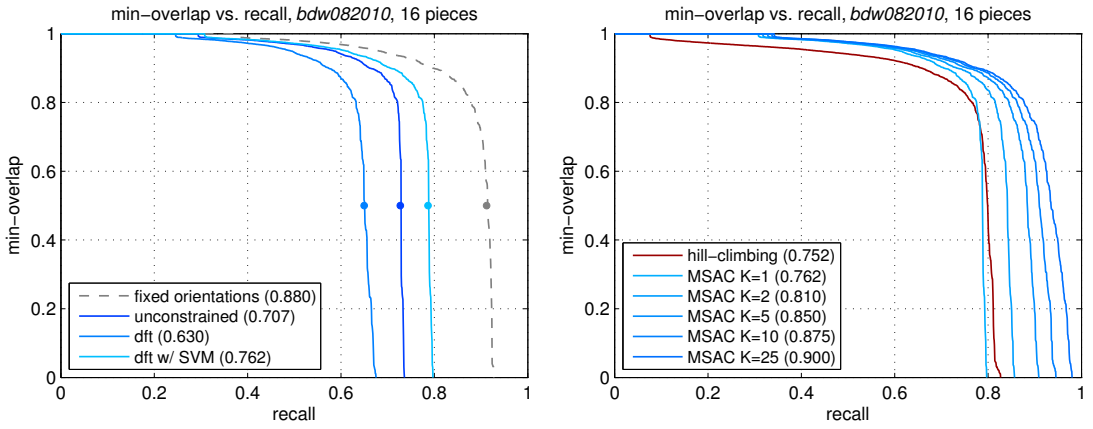
8.7.4 Experiments

We now evaluate different strategies for the computation of hypotheses:

Baseline (fixed orientations). As baseline for our experiments we compute the best possible hypothesis by treating the upright direction of all pieces as known and fixed. Since the ground truth specifies how to adjust the orientation of pieces properly, one only needs to determine an optimal translation for the computation of h.

Unconstrained. The second strategy is called "unconstrained" because we do not reject any hypotheses based on their orientation estimates. That is, we effectively disable the first rejection step in section 8.5.1 by setting $T = +\infty$. As illustrated by the broken circle in figure 8.4b, not using the orientation estimates comes down to considering all relative orientations between pieces. Although this approach ensures that the correct hypothesis can not be falsely discarded by our SVM, this strategy is also computationally much more expensive.

Use Estimates from DFT. To accelerate the computation of h, the key idea is to discard transformations for which the relative orientation between pieces is inconsistent with their orientation estimate. By setting $T = -\infty$ in the rejection step in section 8.5.1 we utilize all estimates, which speeds up MSAC significantly due to the large number of hypotheses that can be invalidated. As illustrated in figure 8.4b, trusting the estimates (modulo $90°$) narrows down the range of accepted rotation angles to only four



(a) different search strategies

(b) non-maximum suppression

Figure 8.9: Left: Evaluation of different MSAC search strategies in terms of overlap-recall (OR) curves, allowing only one hypothesis per piece-pair ($K = 1$). Numbers in brackets represent AUC values. **Right:** Comparison of MSAC with hill-climbing as put forward in chapter 6, depending on the number K of non-maximum suppressed hypotheses. MSAC refers to strategy dft w/ SVM in the left plot.

small intervals. However, this inevitably yields an incorrect result if either estimate is incorrect.

Conditionally Use Estimates from DFT. To get the best of both worlds, we propose to shrink the hypothesis space selectively, e.g., whenever clear lines or text are present on both pieces that give robust orientation estimates. Here we use our SVM model to decide whether or not the estimates are reliable, i.e., the confidence in both estimates is higher than $T = 0.4$. If this is the case, hypotheses are rejected based on their minimal relative error. On the other hand, if one or both of these estimates are unreliable, we retain all hypotheses and perform an unconstrained search. Most often this is the case when pieces have no content or else depict natural images without any clear lines.

Conclusions and Results. In figure 8.9 we show the plots of the overlap-recall curves for our different search strategies. For our experiments we used all strongly adjacent piece-pairs from {test} of the 16 piece version of the *bdw082010* dataset. We note that an unconstrained search gives better results than blindly relying on estimates from the Fourier transform (dft). However, using conditional estimates (dft w/ svm) improves the area under curve from 0.707 to 0.762, at a maximum recall of 0.797. Since the dataset contains a substantial number of pieces that lack text and straight lines, assuming reliable orientation estimates for those pieces would be an overly idealized assumption. In these situations our SVM proves to be very reliable in deciding whether

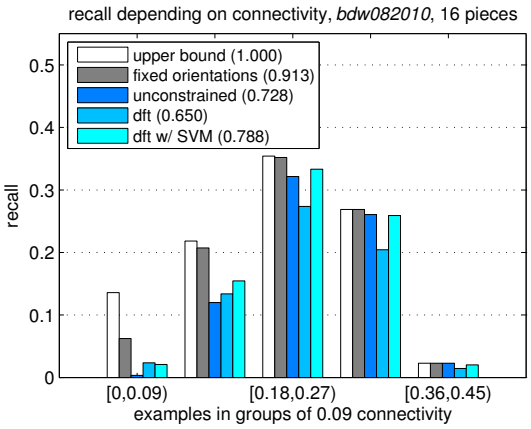


Figure 8.10: Evaluation of recall as a function of connectivity between pieces. Here a prediction counts towards recall (i.e., as true positive) if the overlap is at least 0.5. Recall values (in brackets) are thus associated with the dots in left plot in figure 8.9.

or not to shrink the hypothesis space, depending on the pieces at hand.

The plot on the righthand side in the figure reports the performance for conditionally used estimates depending on the number non-maximum suppressed hypotheses K. Throughout the rest of this thesis we refer to this strategy briefly as "MSAC". As can be seen, even for a moderate increase of K from 1 to 5, the AUC substantially improves from 0.762 to 0.850. Although this trend continues while further increasing K, the effect rapidly diminishes for larger values. This is plausible because for some piece-pairs false positive predictions regarding the orientation estimates – aside from the other rejection criteria – prevent us from reaching the maximum recall of 1.0. It also becomes apparent from this plot that for $K = 1$, MSAC accomplishes essentially identical performance as compared to our hill-climbing approach discussed in chapter 6.

In a second experiment we evaluate the dependency of recall on the adjacency of piece-pairs (i.e., the pieces' max-connectivity). As mentioned before, we expect that correctly aligning pieces is inherently more difficult for piece-pairs with low connectivity, as opposed to those with high connectivity. This claim is now substantiated by the plot in figure 8.10, which shows that our alignment mostly fails when facing low-connectivity examples ($0\%-18\%$). In this scenario, using the SVM clearly improves the recall over an unconstrained search. For high-connectivity examples ($18\% - 45\%$), the SVM still performs on a par, despite being much faster due to the inherently smaller hypothesis space.

In our final experiment we evaluate the number of hypotheses that are retained af-

	$E[\|Q\|]$	$\|\mathcal{H}^{(0)}\|$	$\|\mathcal{H}^{(1)}\|$	$\|\mathcal{H}^{(2)}\|$	$\|\mathcal{H}^{(3)}\|$	$\|\mathcal{H}^{(4)}\|$	$\|\mathcal{H}^{(5)}\|$	$\|\mathcal{H}^{(6)}\|$
mean	65,902	66,716	22,993	9,760	1,965	1,292	885	1
standard deviation	38,916	37,908	39,129	15,489	2,434	1,553	1,060	0

Table 8.1: Number of hypotheses retained after each stage of our variant of MSAC. Our estimate $E[|Q|]$ (see eq. (8.10)) proves to be a very reliable estimate for the true number of initial hypotheses $|\mathcal{H}^{(0)}|$.

ter each stage of MSAC. The results are summarized in table 8.1. The first observation is tied to our estimate about how many hypotheses are to be expected from running algorithm `cmp-candidate-set` once for every pair of anchor points. In the table we report mean values computed from all piece-pairs. As can be seen, on average $E[|Q|]$ underestimates the number of hypotheses in $\mathcal{H}^{(0)}$ only by about 1.2%. For the computation of $E[|Q|]$ as defined in eq. (8.10) we set the number of support points to $n = \frac{1}{2}(n(k) + n(l))$ and define c as the average circumference of the respective pieces. Beside that we notice that our orientation estimates allows us to reject almost two-third of all hypotheses.

Another observation is that, although the average number of hypotheses drops from $66{,}716$ to $22{,}993$ as result of the first rejection step, the standard deviation remains very high. The reason for this effect is clear: Since we use orientation estimates only conditionally, some piece-pairs have considerably fewer hypotheses than others.

8.8 Summary

In this chapter we have proposed a novel variant of MSAC tailored specifically to the alignment of document pieces. Instead of using random sampling as in RANSAC, we argued that it is more efficient to determine pairs of inlier candidates from contour-based distances. Given these candidates we computed different spatial configurations of pieces, which were subject to verification by a very efficient sequence of rejection and selection steps. For the purpose of non-maximum suppression we also demonstrated how an overlap criterion can be defined in analogy with many other computer vision applications. Furthermore, we discussed how this criterion can be used for a quantitative evaluation. We thoroughly evaluated our proposed method and showed that it performs on par in direct comparison with our hill-climbing approach and even yields a substantially higher recall when using non-maximum suppression.

Chapter 9

Structural Compatibility Model

9.1 Motivation

In this chapter we develop a structured output model in order to assess the compatibility of aligned pieces. The key idea is as follows: Any given spatial configuration imposes *costs* for the matching of pieces along their adjacent boundary regions. Intuitively, if two pieces are aligned correctly, these regions should be represented through two very similar polygonal chains. From a geometric perspective we thus expect many adjacent support points across the two chains. On the other hand, we may also assume that the two pieces show similar content along these boundaries. This continuity can be characterized by the same set of content-based image features previously used for our binary approach. However, in contrast to a binary classification setting, we now take advantage of the structural information available: Instead of considering only a *single* point-pair at a time, we define a *cost model* for *sequences* of point-pairs. Using that model allows us to assess the compatibility score of each spatial configuration, which in turn induces a ranking among all aligned piece-pairs. Spatial configurations with a high compatibility score very often resemble the pieces' spatial arrangement in the original document. Providing ranked lists to human experts could greatly benefit them in a semi-automated reconstruction as only the top ranked proposals need to be reviewed. On the other hand, scores obtained from our cost model can also be utilized for fully automated reconstruction, as will be discussed later in chapter 10.

Parts of the approach presented in this chapter have been discussed in previous work [44]. The remainder of this chapter is organized as follows: First in section 9.2 we give a brief overview on related work. Afterwards, we discuss the fundamentals of structural support vector machines in section 9.3 and give a problem definition in sec-

tion 9.4. In section 9.5 we then introduce our discriminative cost model, which enables the prediction of sequences of matching candidates. We then explain the training and briefly sketch how dynamic programming is used for the efficient inference of optimal sequences. Based on the ranking induced by the cost model we perform an extensive evaluation in terms of average precision, which we explain in section 9.6. The chapter is finally concluded with a short summary in section 9.7.

9.2 Related Work

Structured output prediction is an umbrella term for a supervised learning paradigm which generalizes the notion of binary classifiers. The purpose of structural learning is to overcome the limitation to binary outputs such as $+1$ and -1. Instead, the output may involve more complex *structured objects* such as trees, sequences, or sets. Since the type of the output can be chosen depending on the application, structural learning is widely applicable in various research domains, including, for instance, computer vision, natural language processing, as well as computational biology. An example from the latter domain is the work of Yu et al. [65], which deals with the problem of sequence to structure alignment for protein structures. The authors propose to adopt a structural SVM to overcome difficulties when having to learn complex alignment models with hundred of thousands of features. Both the design of our discriminative model and the inference algorithm employed in this chapter were in parts inspired by that work.

Our approach to structured output prediction is based on the framework of structural SVMs [54] in which the authors address the problem of learning functional dependencies between arbitrary input and output spaces. To accomplish this, they propose a generalized notion of separation margins – a well-known concept, for instance, adopted in SVMs for binary classification – and derive a maximum-margin formulation also for the structured case.

Since structured output spaces are by no means limited to certain kinds of applications, it comes at no surprise that structural SVMs have been applied in many different domains within the last decade. For instance, in the computer vision community Blaschko and Lampert [6] were among the first to utilize structured output prediction for the purpose of object detection in images. Instead of treating the problem traditionally as a binary classification task in conjunction with a sliding window approach, they interpret object localization as the problem of predicting bounding boxes as output of the classifier. From this perspective, the problem becomes inherently "structured" in

the sense that the classifier predicts image coordinates rather than binary labels. Probably one of the most influential works on object detection is that of Felzenszwalb et al. [15], who propose a maximum-margin approach for discriminative training of deformable part models. This approach has often been employed in related structural problem formulations, such as in [39] for the explicit modeling of occlusion patterns, or in [63] for articulated human pose estimation.

9.3 Fundamentals of Structural Support Vector Machines

A *structural support vector machine* (ssvm) is a supervised learning paradigm which allows one to learn a discriminative model capable of predicting complex structured outputs. For example, in the area of natural language parsing, an ssvm could be used for the prediction of a parse tree, provided with any sentence as the input. For this example, given a set of annotated training examples, the aim is to learn an accurate mapping $f : \mathcal{X} \longrightarrow \mathcal{Y}$, where $\mathcal{X}$ is the space of all sentences, and $\mathcal{Y}$ is the structured output domain containing all possible parse trees.

We now give a short introduction to structural support vector machines, which for the most part is based on the discussion in [32]. As for all supervised learning tasks, to learn the mapping f we assume a set of labeled training examples:

$$S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\} \in (\mathcal{X} \times \mathcal{Y})^n \tag{9.1}$$

This set contains n input-output pairs which can be interpreted as a collection of random variables that are independent and identically distributed (i.i.d.) according to some (typically unknown) distribution $P(\boldsymbol{x}, \boldsymbol{y})$. In theory, one would now want to find the best mapping f to minimize the prediction error over all potential instances sampled from P. That is, the aim is to minimize the *risk* as a function of f:

$$R_P^{\Delta}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \Delta(\boldsymbol{y}, f(\boldsymbol{x}))dP(\boldsymbol{x}, \boldsymbol{y}) \tag{9.2}$$

The risk, which amounts to the *expected loss* over all instances $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$, is commonly expressed in terms of a Riemann-Stieltjes integral[1]. Besides, the risk relies on the loss function[2] $\Delta(\boldsymbol{y}, \bar{\boldsymbol{y}})$ to quantify the degree of error for predicting $\bar{\boldsymbol{y}}$ if actually $\boldsymbol{y}$ was the correct choice. Although the choice of an appropriate loss function strongly

[1] This formulation encapsulates the cumulative distribution by means of $dP(\boldsymbol{x}, \boldsymbol{y})$, which differs from the usual Riemann integral only in the case when P is not differentiable, e.g., when the output space $\mathcal{Y}$ is discrete.
[2] Note that the loss function is generally non-convex and can be even discontinuous.

depends on the learning problem, it is common practice to assume $\Delta(\boldsymbol{y}, \boldsymbol{y}) = 0$ and $\Delta(\boldsymbol{y}, \bar{\boldsymbol{y}}) \geq 0$ for $\bar{\boldsymbol{y}} \neq \boldsymbol{y}$. In practice we concentrate on $\boldsymbol{w}$-parameterized model classes, which lets us express the mapping function by $f_{\boldsymbol{w}}$. The problem with learning parameters $\boldsymbol{w}$ is that the distribution P is typically not known. To circumvent this problem, it is put forward in [32] to follow the Empirical Risk Minimization Principle [57], which means that our mapping $f_{\boldsymbol{w}}$ is inferred solely from the training set. Since our training examples in S are sampled from P and assumed to be i.i.d., one can assess the quality of $f_{\boldsymbol{w}}$ in terms of the *empirical risk* on training set S:

$$R_S^{\Delta}(f_{\boldsymbol{w}}) = \frac{1}{n} \sum_{i=1}^{n} \Delta(\boldsymbol{y}_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \tag{9.3}$$

In order to avoid an overfitting of model $\boldsymbol{w}$ to the training examples, one typically minimizes the *regularized empirical risk* instead. For a definition of the actual optimization problem, we further require a linear discriminant function $\Omega_{\boldsymbol{w}} : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathbb{R}$. The purpose of this function is to measure the compatibility of any possible output $\bar{\boldsymbol{y}}$ for input $\boldsymbol{x}$, regarding model $\boldsymbol{w}$. In its most general form, the optimization problem can then be written as follows:

$$\begin{aligned} &\underset{\boldsymbol{w}, \xi_i \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i \right\} \\ &\text{s.t.} \quad \forall \bar{\boldsymbol{y}}_i \in \mathcal{Y} \; : \; \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \geq \Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i) - \xi_i \end{aligned} \tag{9.4}$$

The above formulation is a convex quadratic optimization problem, which is commonly called the n-slack soft-margin formulation with margin-rescaling. The objective function incorporates a regularization term $\frac{1}{2} \|\boldsymbol{w}\|^2$, which safeguards the model from overfitting. Since a smaller ℓ_2-norm corresponds to a broader margin for the separation of examples, the problem is regarded as a "maximum-margin" approach. Due to the structured output space $\mathcal{Y}$ (e.g., the set of all possible parse trees), the optimization problem involves a potentially very large number of constraints for each training example:

$$\forall \bar{\boldsymbol{y}}_i \in \mathcal{Y} \; : \; \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \geq \Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i) - \xi_i \tag{9.5}$$

This set of constraints for the i-th training example states that any possible output $\bar{\boldsymbol{y}}_i \in \mathcal{Y}$ has to be assigned a score by model $\boldsymbol{w}$ which is at least $\Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i)$ lower than that of the correct output $\boldsymbol{y}_i$. Violations of these constraints are expressed in terms of *slack variables* ξ_i, which are penalized in the objective function. Due to the structured nature of output space $\mathcal{Y}$, the problem typically involves a very large number of constraints.

This seemingly makes the optimization problem intractable to solve. However, it can in fact be solved efficiently (e.g., by a cutting-plane algorithm, or approximately by stochastic gradient descent). The reason lies in the fact that the optimization problem involves only n margin violations. That is, all constraints regarding a given training example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ are subsumed into one slack variable ξ_i. This becomes more apparent when solving the inequalities in the constraints in eq. (9.4) for slack variable ξ_i, which represents the margin violation of the i-th training example:

$$\xi_i \geq \max_{\bar{\boldsymbol{y}}_i \in \mathcal{Y}} \left\{ \Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i) - \left(\Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \right) \right\} \tag{9.6}$$

An obvious lower-bound on ξ_i is 0, which immediately results from substituting $\bar{\boldsymbol{y}}_i = \boldsymbol{y}_i$ in eq. (9.6). Besides, the slack variable is also 0 whenever the prediction for the correct output is higher than that for any incorrect output $\bar{\boldsymbol{y}}_i \neq \boldsymbol{y}_i$, by at least $\Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i)$. That is, there is no training error for the i-th example if the separation margin $\Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \geq \Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i)$ is sufficiently large for each output $\bar{\boldsymbol{y}}_i \in \mathcal{Y}$.

One question still remains to be answered: Why would we minimize the sum over slack variables, if in fact our goal is to minimize the empirical risk? The answer is that direct minimization of the empirical risk is generally infeasible because the loss function is typically neither convex nor continuous. To make the optimization tractable, one replaces each loss value in the empirical risk (eq. (9.3)) with its corresponding slack variable (eq. (9.6)). This applies a convex upper bound on the incurred losses[1] as each slack variable can be seen as putting a hinge-loss style bound on the loss. Using $\hat{\boldsymbol{y}}(\boldsymbol{w}) = f_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname{argmax}_{\bar{\boldsymbol{y}} \in \mathcal{Y}} \Omega_{\boldsymbol{w}}(\boldsymbol{x}, \bar{\boldsymbol{y}})$ as the mapping function, the upper bound on the empirical risk is obtained by:

$$\frac{1}{n} \sum_{i=1}^{n} \xi_i \geq \frac{1}{n} \sum_{i=1}^{n} \left[\max_{\bar{\boldsymbol{y}}_i \in \mathcal{Y}} \left\{ \Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i) + \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \right\} - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right] \tag{9.7}$$

$$\geq \frac{1}{n} \sum_{i=1}^{n} \left[\Delta(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i(\boldsymbol{w})) + \max_{\bar{\boldsymbol{y}}_i \in \mathcal{Y}} \left\{ \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \bar{\boldsymbol{y}}_i) \right\} - \Omega_{\boldsymbol{w}}(\boldsymbol{x}_i, \boldsymbol{y}_i) \right] \tag{9.8}$$

$$\geq \frac{1}{n} \sum_{i=1}^{n} \Delta(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i(\boldsymbol{w})) = \frac{1}{n} \sum_{i=1}^{n} \Delta(\boldsymbol{y}_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i)) \tag{9.9}$$

We briefly discuss the steps of this derivation in reverse order: In the bounding step from (9.9) to (9.8) we add a non-negative term, which corresponds to the largest margin violation. The second step from (9.8) to (9.7) removes the implicit dependency

[1] For a thorough discussion on bounds for structured output prediction please refer to [9, 54].

of the loss function on model $\boldsymbol{w}$ that exists through $\hat{\boldsymbol{y}}_i(\boldsymbol{w})$. Reorganizing terms in (9.7) makes clear that each summand is identical to the righthand side of eq. (9.6). Convexity of this bound in $\boldsymbol{w}$ follows from the fact that we only use a sum and a maximum operation over the discriminant function $\Omega_{\boldsymbol{w}}$ that is linear in $\boldsymbol{w}$.

9.4 Problem Definition

Our problem formulation considers instances from input space $\mathcal{X}$, which are given by piece-pair $\boldsymbol{x} = (\mathcal{P}_k, \mathcal{P}_l)$. The joint output space $\mathcal{H} \times \mathcal{Y}$ consists of two subspaces. The first subspace consists of hypotheses for rigid transformations that align the two pieces. Each hypothesis $h \in \mathcal{H}$ within this set entails a pair of adjacent boundary regions $\mathcal{B}(h; \boldsymbol{x}) = (B_k(h), B_l(h))$. We discussed in the last chapter (see section 8.5.5) how these regions are determined and how they can be used for non-maximum suppression and evaluation. The second subspace $\mathcal{Y}$ contains sequences of match and gap operations, which reflect all possibilities how support points from one piece can be associated with points from the other piece. We represent these sequences $\boldsymbol{y} \in \mathcal{Y}$ as a list of operations with variable length. In practice we only consider point-correspondences between the pieces' adjacent boundary regions, because points from one piece that are spatially disconnected from the other piece are uninformative for the correctness of the spatial configuration. Thus, all possible sets of point-correspondences identified through match operations are summarized by $\mathfrak{P}\big(B_k(h) \times B_l(h)\big)$, where $\mathfrak{P}$ represents the power set and $B_k(h) \times B_l(h)$ is the Cartesian product of points within the adjacent boundary regions entailed by h.

In practice, given any piece-pair $\boldsymbol{x}$, our goal is to predict the best transformation $\hat{h}$, together with its optimal sequence $\hat{\boldsymbol{y}}$. Assuming we have already trained a model $\boldsymbol{w}$, we can formalize this idea by introducing *decision function* $f_{\boldsymbol{w}} : \mathcal{X} \longrightarrow \mathcal{H} \times \mathcal{Y}$, which determines the structured output $f_{\boldsymbol{w}}(\boldsymbol{x}) = (\hat{h}, \hat{\boldsymbol{y}})$ according to:

$$(\hat{h}, \hat{\boldsymbol{y}}) = \underset{(h,\boldsymbol{y}) \in \mathcal{H} \times \mathcal{Y}}{\operatorname{argmax}} \big\{ \Omega_{\boldsymbol{w}}\big(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})\big) \big\} \tag{9.10}$$

For this joint inference task we utilize a *linear discriminant function* $\Omega_{\boldsymbol{w}} : \mathcal{X} \times \mathcal{H} \times \mathcal{Y} \to \mathbb{R}$ defined by:

$$\Omega_{\boldsymbol{w}}\big(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})\big) = \langle \boldsymbol{w}, \boldsymbol{\Psi}(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})) \rangle \tag{9.11}$$

Our definition makes use of $\boldsymbol{\Psi}(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x}))$, which is a *joint feature map* defined over input space $\mathcal{X}$ and output space $\mathcal{H} \times \mathcal{Y}$. As will be discussed shortly, this joint feature map is a fixed size representation which describes any given sequence of point-

correspondences in terms of geometric and content-based features. Note that it depends on the piece-pair $\boldsymbol{x} \in \mathcal{X}$ and transformation $h \in \mathcal{H}$ only indirectly through the entailed adjacent boundary regions $\mathcal{B}(h; \boldsymbol{x})$ because it is those two regions that define the scope of the sequence. However, it directly depends on the sequence $\boldsymbol{y}$ because its match and gap operations characterize the pieces' compatibility along $\mathcal{B}(h; \boldsymbol{x})$.

9.4.1 Implementation Requirements

Solving the aforementioned joint inference task requires an efficient implementation of the following three components:

$$\mathcal{B}(h; \boldsymbol{x}) \tag{9.12}$$

$$\boldsymbol{\Psi}(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})) \tag{9.13}$$

$$(\hat{h}, \hat{\boldsymbol{y}}) = \underset{(h, \boldsymbol{y}) \in \mathcal{H} \times \mathcal{Y}}{\operatorname{argmax}} \left\{ \Omega_{\boldsymbol{w}}\big(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})\big) \right\} \tag{9.14}$$

First, after positioning pieces in $\boldsymbol{x}$ regarding hypothesis h, one needs to identify the pieces' adjacent boundary regions $\mathcal{B}(h; \boldsymbol{x})$. We discussed how this problem can be solved in the previous chapter in section 8.5.5. With respect to those boundary regions we then have to define a joint feature map $\boldsymbol{\Psi}(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x}))$ that allows us to represent any possible sequence $\boldsymbol{y}$ of matching candidates. Finally, we have to come up with an algorithm to find the optimal sequence $\hat{\boldsymbol{y}}$ together with its transformation $\hat{h}$ regarding the cost model $\boldsymbol{w}$.

Note that changes to h lead to a different spatial configuration, which in turn affects the two adjacent boundary regions $\mathcal{B}(h; \boldsymbol{x}) = (B_k(h), B_l(h))$. To make the joint inference task in eq. (9.14) tractable, we relax the original problem formulation. That is, we perform approximate inference by treating each subspace individually: First, we generate a small but diverse set of hypotheses using MSAC with non-maximum suppression (see chapter 8). For each of these hypotheses h we can limit the scope of sequences to point-pairs across the two regions in $\mathcal{B}(h; \boldsymbol{x})$. Then, separately for each hypothesis, we determine the best sequence. The pair $(\hat{h}, \hat{\boldsymbol{y}})$ with the highest discriminant function value is finally chosen as the approximate solution.

9.5 Linear Model

In this section we adopt the idea of Yu et al. [65], who used a structural SVM for sequence to structure alignment in protein models. One of the ideas put forward in that

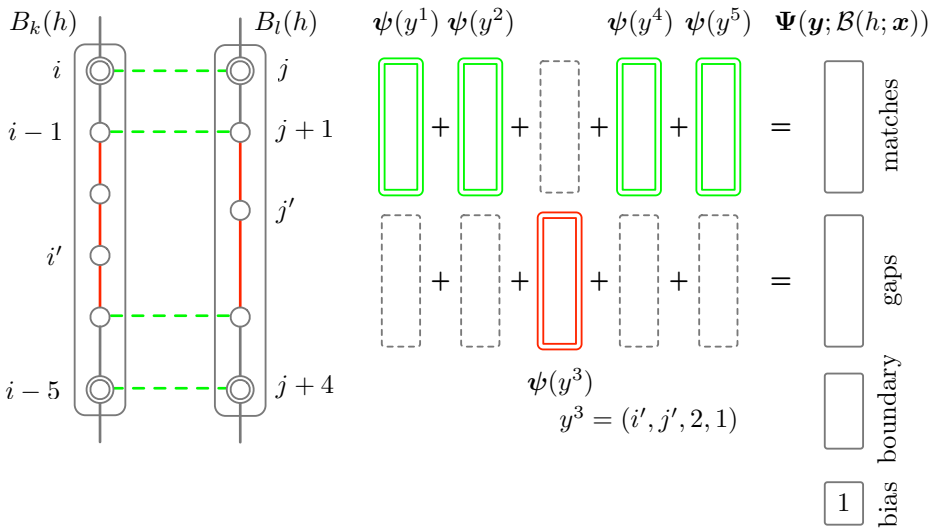


Figure 9.1: Left: Schematic illustration of a sequence of length five, with four match operations (green). In between, gap y^3 (red) spans over two support points on $B_k(h)$ and one point on $B_l(h)$, respectively. The descriptors for individual operations ψ are part of the sequence descriptor $\boldsymbol{\Psi}$ (eq. (9.15)), as illustrated on the righthand side.

work is to decompose sequences in additive manner. This particular aspect can easily be transferred to our application: Instead of considering the cost incurred for the substitution of two amino acids, we aim to quantify how well two matching candidates across adjacent boundaries fit. In our problem setting, a sequence $\boldsymbol{y} = (y^1, \ldots, y^{|\boldsymbol{y}|})$ is defined as a variable number of match and gap operations. In the ideal case, the sequence identifies all and only inliers across the pieces by means of match operations. Intuitively, each such *match operation* associates one support point from piece $\mathcal{P}_k$ with a second one on $\mathcal{P}_l$. For instance, we write $y^m = (i', j', 0, 0)$ to associate support point $i' \in B_k(h)$ with $j' \in B_l(h)$. The latter two 0's indicate that a match does not affect any points other than i' and j'. Ideally, we only associate i' and j' with each other through a match if those two support points were adjacent in the original document (i.e., an inlier). However, one problem is that even for inliers, image contents on both pieces may be incompatible if the tearing boundary runs between two distinct content elements (e.g., in between text and a table with different background color). Furthermore, since we are dealing with real-world documents, our pieces do not always come with only well aligned support points.

To account for this fact we complement matches by *gap operations*. A gap allows us to skip (possibly multiple) support points on either of the two pieces. We formalize this idea by $y^m = (i', j', \delta_k, \delta_l)$, which introduces gaps at points i' and j' spanning δ_k

and δ_l many points, respectively. An illustrating example is given in figure 9.1, which shows a sequence of length five. In this example, the sequence consists of four match operations that identify point-correspondences between the pieces as well as a single gap, which for instance can be caused by noise in the contour approximation.

9.5.1 Decomposition of Sequences into Operations

Next we define a descriptor for sequences that is linear in the individual operations. As in [65] we decompose the descriptor for the entire sequence into the sum over all descriptors of individual operations:

$$\boldsymbol{\Psi}(\boldsymbol{y}; \mathcal{B}(h; \boldsymbol{x})) = \left(1, \boldsymbol{\psi}_{boundary}(\mathcal{B}(h; \boldsymbol{x})), \sum_{m=1}^{|\boldsymbol{y}|} \boldsymbol{\psi}_{op}(y^m)\right) \in \mathbb{R}^{1+2+d} \tag{9.15}$$

The descriptor consists of three parts: (i) A constant term 1 which enables learning of a global bias, (ii) a descriptor to characterize the dissimilarity of the two adjacent boundary regions, as well as (iii) the sum over descriptors of individual operations. The first descriptor is defined by:

$$\boldsymbol{\psi}_{boundary}(\mathcal{B}(h; \boldsymbol{x})) = (-1/\lfloor r_k, r_l \rfloor, -1 + \lfloor r_k, r_l \rfloor / \lceil r_k, r_l \rceil) \tag{9.16}$$

The first term is chosen inversely proportional to the minimum of the length of the two boundary segments, denoted by $r_k = |B_k(h)|$ and $r_l = |B_l(h)|$. This effectively devalues spatial configurations constituting only short adjacent boundary regions. The second term is a penalty for boundary regions that are uneven in length in terms of the number of support points.

Since each operation can either be a match or a gap, it is possible to split the second descriptor from eq. (9.15) into blocks. We write $\boldsymbol{\psi}_{op}(y^m) = \big(\boldsymbol{\psi}_{match}(y^m), \boldsymbol{\psi}_{gap}(y^m)\big)$, which denotes the concatenation of the individual match and gap descriptor. Note that depending on the operation, exactly one of the two descriptors is set while the other one is initialized to $\mathbf{0}$. Each descriptor reflects the compatibility of only those parts of $B_k(h)$ and $B_l(h)$ that are affected by the respective operation, as will be discussed shortly. In analogy to the descriptor we can decompose our model into blocks, which gives:

$$\boldsymbol{w} = (w_{bias}, \boldsymbol{w}_{boundary}, \boldsymbol{w}_{match}, \boldsymbol{w}_{gap}) \tag{9.17}$$

Note that all components of our model, except for the scalar w_{bias}, are constrained to non-negative values in order to retain the notion of costs. This will become clear in

conjunction with our operations which are discussed next:

Match Operation. The descriptor for a match operation $y^m = (i', j', 0, 0)$ describes the compatibility of support point i' on $B_k(h)$ with j' on $B_l(h)$. In eq. (9.18) the first two elements introduce an absolute and a relative offset. The first term is a constant 1, which enables learning of a bias for making a match. The latter is the reciprocal value of the minimum over the number of support points $r_k = |B_k(h)|$ and $r_l = |B_l(h)|$. This effectively rewards matches between short regions more than for long ones.

The second component $\boldsymbol{d}_{match}(y^m)$ makes use of the dissimilarities introduced in chapter 4 (see eq. 4.1) [1]. Each of these dissimilarity values stems from one of our geometric and content-based features that were also used for binary classification in chapter 5. In addition to those feature we here also use the points' spatial proximity after alignment through a hypothesis h. Our *match descriptor* is given by:

$$\boldsymbol{\psi}_{match}(y^m) = \left(1, 1/\lfloor r_k, r_l \rfloor, -\boldsymbol{d}_{match}(y^m)\right) \tag{9.18}$$

Note that only the first two terms in our descriptor are non-negative. Thus, the dot product of model $\boldsymbol{w}_{match}$ and descriptor $\boldsymbol{\psi}_{match}(y^m)$ amounts to the cost incurred for making a match, which is offset only by the bias and the compatibility of the adjacent boundary regions.

Gap Operation. The introduction of gaps enables the model to deal with incompatible image contents and noise in the contour approximation. The principle idea is that each gap operation $y^m = (i', j', \delta_k, \delta_l)$ is delimited by two matches. For example, the gap y^3 in figure 9.1 is immediately followed by a match $y^{m+1} = (i' - 1, j' + 1, 0, 0)$, and it is preceded by another match $y^{m-1} = (i' + \delta_k, j' - \delta_l, 0, 0)$, with $\delta_k = 2$ and $\delta_l = 1$. Analogous to a match, the first two terms in eq. (9.19) introduce offsets to learn a flat penalty. Since gaps should have a negative contribution to the score of a sequence, those offsets have a negative sign.

The second component $\boldsymbol{d}_{gap}$ is designed to penalize long gaps and those that are uneven in size (in terms of δ_k and δ_l). To accomplish this, we add two dissimilarity values $\delta_k + \delta_l$ and $\lceil \delta_k, \delta_l \rceil - \lfloor \delta_k, \delta_l \rfloor$ in $\boldsymbol{d}_{gap}$. Furthermore, we also add dissimilarities that encode how much the pieces' polygons diverge along the gap regions. Our final *gap descriptor* is obtained by stacking the dissimilarity vector onto the gap penalty offsets:

$$\boldsymbol{\psi}_{gap}(y^m) = \left(-1, -1/\lfloor r_k, r_l \rfloor, -\boldsymbol{d}_{gap}(y^m)\right) \tag{9.19}$$

[1] We do not use the shape feature here because pieces are already aligned. All content-based features use LCE segments with 40 pixels in length, just like for all experiments with our binary approach.

9.5.2 Learning Problem

As discussed earlier, learning the cost model $\boldsymbol{w}$ is treated as regularized empirical risk minimization problem. Our training set $S = \{(\boldsymbol{x}_1, \boldsymbol{z}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{z}_N)\}$ consists of N piece-pairs represented by $\boldsymbol{x}_i = (\mathcal{P}_k, \mathcal{P}_l)$, as well as their annotations $\boldsymbol{z}_i = (l_i, h_i, \boldsymbol{y}_i)$. Labels $l_i \in \{-1, +1\}$ distinguish positive from negative examples, for which we use all degrees of fragmentation (8, 16, 24, and 32 pieces per page) of the *bdw082010* training datasplit. Our positive examples stem from pairs of pieces that are strongly adjacent in the manually reconstructed document page (i.e., piece-pairs with 4 or more inliers). For each such pair, we use our ground truth to determine the optimal transformation h_i as well as the correct sequence $\boldsymbol{y}_i \neq \mathbf{0}$. The negative examples on the other hand are limited to sequences among non-adjacent piece-pairs of the same page. A natural choice to indicate that there exists no correct sequence is to define $\boldsymbol{y}_i = \emptyset$, in which case the joint feature map is set to $\boldsymbol{\Psi}(\boldsymbol{y}_i; \mathcal{B}(h_i; \boldsymbol{x}_i)) = \mathbf{0}$.

Using the 0-1 loss function, we obtain a cost model by solving the following optimization problem (OPT):

$$\begin{aligned} &\underset{\boldsymbol{w}, \xi_i \geq 0}{\operatorname{argmin}} \left\{ \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \xi_i \right\} \\ \text{s.t.} \quad &\forall i : l_i = +1 \;:\; \Omega_{\boldsymbol{w}}\big(\boldsymbol{y}_i; \mathcal{B}(h_i; \boldsymbol{x}_i)\big) \geq +1 - \xi_i \\ &\forall i : l_i = -1,\ \forall (\bar{h}_i, \bar{\boldsymbol{y}}_i) : \; \Omega_{\boldsymbol{w}}\big(\bar{\boldsymbol{y}}_i; \mathcal{B}(\bar{h}_i; \boldsymbol{x}_i)\big) \leq -1 + \xi_i \\ &\boldsymbol{w}_{boundary}, \boldsymbol{w}_{match}, \boldsymbol{w}_{gap} \geq \mathbf{0} \end{aligned} \tag{9.20}$$

We additionally impose non-negativity constraints on all blocks of the model (except for the global bias w_{bias}) in order to enforce the notion of costs. Apart from that, the above problem formulation is a special case of the convex optimization problem described before in OPT 9.4.

To see that both optimization problems are conceptually the same, it is important to note that we explicitly split the training set into a positive and negative subset. For each positive example an empty sequences $\bar{\boldsymbol{y}}_i = \emptyset$ is considered as the only possible incorrect prediction. Consequently, there is only a single constraint for these examples. Our choice for $\Delta(\boldsymbol{y}_i, \bar{\boldsymbol{y}}_i)$ is the 0-1 loss function, which returns 1 because $\boldsymbol{y}_i \neq \bar{\boldsymbol{y}}_i = \emptyset$. By substituting our definition $\boldsymbol{\Psi}(\emptyset; \mathcal{B}(h_i; \boldsymbol{x}_i)) = \mathbf{0}$ into the discriminant function in OPT 9.4 we obtain the constraints on positive examples used above. One can proceed analogously for the constraints on the subset of negative examples, i.e., by substitution and re-arrangement of terms. The only difference here is that $\bar{\boldsymbol{y}}_i \neq \boldsymbol{y}_i = \emptyset$.

A direct consequence of using the 0-1 loss is that all sequences from adjacent piece-pairs are constrained to score $+1$ or higher. On the other hand, any sequence across non-adjacent pieces must score less than or equal -1. Theoretically, the above problem formulation requires examining all possible spatial configurations for each non-adjacent piece-pair. However, this would clearly make the training impracticable. To circumvent this problem, we use our MSAC approach once in advance to determine a diverse set of $K = 10$ non-maximum suppressed hypotheses. We want to emphasize that this is sufficient to provide a substantial number of negative examples, because any of these configurations involves plenty of possible sequences.

In spite of the fact that this formulation establishes multiple constraints on each non-adjacent piece-pair $\boldsymbol{x}_i$, only the strongest of all margin violations yields the non-negative value for slack variable ξ_i. The sum over all slacks provides an upper bound on the number of training errors[1], which is subject to minimization and is thus used in the second term of the objective function. For all of our experiments we choose parameter $C = 1$ to balance the training error versus the model's capability of generalizing beyond the training examples.

To obtain the solution $\boldsymbol{w}$ for our convex optimization problem, we use the stochastic gradient descent (SGD) solver of Felzenszwalb et al. [15]. In practice we perform multiple rounds of hard-negative mining to build a cache of sequences from negative examples while retaining the one fixed sequence provided by each positive example. After each round, all negative examples with zero margin violation are removed from the cache. The training continues as long as a sufficient number of new hard negative examples can be found that were not previously added to the cache.

9.5.3 Stochastic Gradient Descent

To show how the optimization problem in OPT 9.20 can be solved, we first re-arrange the constraints for ξ_i and substitute the result in the objective function. This leads to the following alternative problem formulation

$$L(\boldsymbol{w}) = \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{n} \ell_{\boldsymbol{w}}(\hat{\boldsymbol{y}}_i; \hat{h}_i, \boldsymbol{x}_i) \tag{9.21}$$

with $\ell_{\boldsymbol{w}}(\hat{\boldsymbol{y}}_i; \hat{h}_i, \boldsymbol{x}_i) = \left\lceil 0, 1 - l_i \Omega_{\boldsymbol{w}}\big(\hat{\boldsymbol{y}}_i; \mathcal{B}(\hat{h}_i; \boldsymbol{x}_i)\big) \right\rceil$ being the hinge-loss. It is common to use the hinge-loss as a tight convex upper bound for the 0-1 loss function, which itself is non-convex and hence unsuitable for optimization. Although the hinge-loss is only

[1] In our implementation we use C instead of $\frac{C}{n}$ as term for balancing the training error versus model regularization.

piecewise linear and thus non-differentiable[1], we can still compute the sub-gradient with respect to model $\boldsymbol{w}$:

$$\nabla L(\boldsymbol{w}) = \boldsymbol{w} + C \sum_{i=1}^{n} \nabla \ell_{\boldsymbol{w}}(\boldsymbol{x}_i; \hat{h}_i, \hat{\boldsymbol{y}}_i) \tag{9.22}$$

The sub-gradient of the hinge-loss equates to:

$$\nabla \ell_{\boldsymbol{w}}(\boldsymbol{x}_i; \hat{h}_i, \hat{\boldsymbol{y}}_i) = \begin{cases} 0 & \text{if } l_i \Omega_{\boldsymbol{w}}\big(\hat{\boldsymbol{y}}_i; \mathcal{B}(\hat{h}_i; \boldsymbol{x}_i)\big) \geq 1 \\ -l_i \boldsymbol{\Psi}(\hat{\boldsymbol{y}}_i; \mathcal{B}(\hat{h}_i; \boldsymbol{x}_i)) & \text{otherwise} \end{cases} \tag{9.23}$$

In stochastic gradient descent one repeatedly updates the model using only a small subset of examples at a time. Since the hinge-loss provides a convex upper bound on the training error, $L(\boldsymbol{w})$ is a convex function, and thus one can find the optimal solution by successively taking steps in the negative direction of $\nabla \ell_{\boldsymbol{w}}$. In the implementation of [15], which we adopt mostly unmodified for our application, the model is regularized periodically once after a fixed number of training rounds. To retain the notion of costs in $\boldsymbol{w}$ we enforce the lower bounds on the model $\boldsymbol{w}_{boundary}, \boldsymbol{w}_{match}, \boldsymbol{w}_{gap} \geq \mathbf{0}$ after each regularization step.

9.5.4 Inference by Dynamic Programming

Solving the optimization problem involves repeatedly solving the joint inference task. To make this an efficient operation, we use a modified variant of the Smith-Waterman algorithm [50], which allows us to determine the best sequence of match and gap operations across the adjacent boundary regions $\mathcal{B}(h; \boldsymbol{x}) = (B_k(h), B_l(h))$. Recall that these two regions are represented by polygonal chains whose points are arranged according to a linear order induced by their starting points. To simplify the discussion, we ignore the cyclic gap between the first and the last support points on either piece and represent both polygonal chains through index sets $\{i, \ldots, i - r_k + 1\}$ and $\{j, \ldots, j + r_l - 1\}$.

We restrict each sequence to start with a match in (i, j) and end with a match in $(i - r_k + 1, j + r_l - 1)$. In order to perform inference efficiently we can take advantage of our linear model. Since sequences are decomposable into individual operations, one can incrementally extend sequence prefixes into longer sequences by appending only one operation at a time to an existing prefix. Let $p[i', j']$ denote the score of the prefix that starts in (i, j) and ends in (i', j'). In order to compute that prefix's score based on

[1] This is due to the slack variables being constrained to non-negative values $\xi_i \geq 0$.

shorter prefixes, we can either establish a match with score

$$\begin{aligned} p[i', j'] &= p[i' + 1, j' - 1] + \beta) \\ \beta &\leftarrow \langle \boldsymbol{w}_{match}, \boldsymbol{\psi}_{match}(y) \rangle \\ y &\leftarrow (i', j', 0, 0), \end{aligned} \tag{9.24}$$

or else introduce a gap if

$$p[i', j'] = \max_{\delta_k, \delta_l} \Big\{ p[i' + \delta_k, j' - \delta_l] + \beta \Big\} \tag{9.25}$$

$$\begin{aligned} \beta &\leftarrow \langle \boldsymbol{w}_{gap}, \boldsymbol{\psi}_{gap}(y) \rangle \\ y &\leftarrow (i', j', \delta_k, \delta_l) \end{aligned} \tag{9.26}$$

provides a higher score. All prefix scores for the two adjacent boundary regions can be compactly stored in a matrix of size $r_k \times r_l$. Beside that, a second matrix is used to store the last operation that led to any given prefix. To infer the best sequence $\hat{\boldsymbol{y}}$ from the resulting matrices, we then perform a traceback from the last to the first match and accumulate the encountered operations in reverse order in $\hat{\boldsymbol{y}}$. Notice that the bias w_{bias}, as well as the score for the adjacent boundary regions $\langle \boldsymbol{w}_{boundary}, \boldsymbol{\psi}_{boundary}(\mathcal{B}(h; \boldsymbol{x})) \rangle$, can simply be added once in the end as neither of these values depends on the actual choice of $\hat{\boldsymbol{y}}$.

9.6 Evaluation

This section discusses a novel evaluation methodology which regards document reconstruction as a ranking problem. For this purpose, we have to assign a confidence to each aligned piece-pair. This is exactly what structured output prediction gives us: We obtain the score of the best possible sequence $\hat{\boldsymbol{y}}$ regarding model $\boldsymbol{w}$, which we call the *ssvm score*. We use this score as a measure of confidence in prediction $(\hat{h}, \hat{\boldsymbol{y}})$. Based on the ssvm scores we then induce a ranking among all of our aligned piece-pairs. This ranking is *optimal* if all adjacent piece-pairs are identified correctly *and ranked before* the first pair of non-adjacent pieces. Determining the quality of such ranked lists is for instance also important for the evaluation of image retrieval systems, for which it is common to use the *average precision* as the performance measure. The following sections demonstrate how this standard technique can be adapted to our problem domain.

9.6.1 Precision & Recall

The *precision* is a very common performance measure,which is, for instance, used for the evaluation of image retrieval and object detection systems. Intuitively, it quantifies the number of correct predictions (true positives) in relation to the number of all predictions (true positives and false positives). The precision is often used in conjunction with the recall to measure the quality of retrieval or classification results. For the evaluation in this chapter, we define the *recall* as the percentage of strongly adjacent piece-pairs which have been aligned correctly in terms of our min-overlap criterion.

In order to introduce precision and recall formally, we first need to define the terms *true positive* and *false positive*. For now, let us assume that the belonging of pieces to document pages is known and that we are only interested in reconstructing individual document pages[1]. For the discussion here, let us consider piece-pairs of the i-th page, which are indexed by (k, l). First, with respect to a threshold T on the ssvm score, we determine the number of true positives by:

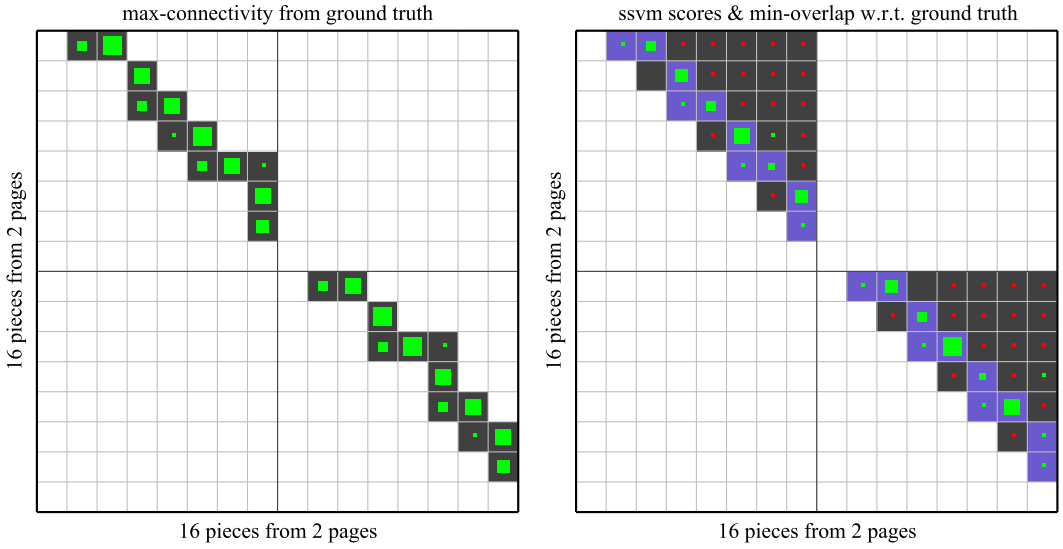
$$TP_i(T) = \sum_{(k,l) \in \text{Page}_i} tp(k, l; T) \tag{9.27}$$

The function that actually decides whether our ssvm provides a correct output for a given example $\boldsymbol{x} = (\mathcal{P}_k, \mathcal{P}_l)$ is defined by:

$$tp(k, l; T) = \big[\Omega_{\boldsymbol{w}}\big(\hat{\boldsymbol{y}}; \mathcal{B}(\hat{h}; \boldsymbol{x})\big) \geq T\big] \big[\Gamma_{k,l}(\hat{h}, h^*) \geq 0.5\big]. \tag{9.28}$$

Here we denote by $[\cdot]$ the Iverson bracket which returns 1 if its boolean condition is satisfied, and 0 otherwise. Recall that $f_{\boldsymbol{w}}(\boldsymbol{x}) = (\hat{h}, \hat{\boldsymbol{y}})$ is the prediction with the highest score regarding our cost model $\boldsymbol{w}$. The first term in eq. (9.28) yields 1 whenever the ssvm score exceeds threshold T. With respect to the ground truth, the second term assesses the correctness of the predicted hypothesis $\hat{h}$. For this purpose we compute the min-overlap $\Gamma_{k,l}(\hat{h}, h^*)$ of the two adjacent boundary regions entailed by prediction $\hat{h}$ and ground truth h^* (see section 8.7). Any prediction can thus count as a true positive only if the min-overlap entailed by $\hat{h}$ exceeds 0.5. Also note that a high min-overlap by itself is not a sufficient condition for a true positive because it also requires a high confidence in our chosen alignment. Consequently, lowering the threshold T leads to more true positives but also increases the number of false positives.

[1] We are going to lift this restriction in the following chapter, where we deal with the simultaneous reconstruction of multiple pages. Both precision and recall are also directly applicable to this more general problem setting.



(a) ground truth information

(b) ssvm predictions

Figure 9.2: Hinton diagrams for 16 pieces stemming from two document pages torn into 8 pieces each. Patch sizes are scaled according to the max-connectivity (left), and ssvm score (right). Green patches represent positive values, red ones indicate negative values. **Left:** Adjacency matrix representing the max-connectivity (ground truth) between piece-pairs within each page. **Right:** ssvm-scores (predictions) for each piece-pair. A cell with purple background indicates a min-overlap of 0.5 or higher.

In similar manner we compute the number of false positives according to:

$$FP_i(T) = \sum_{(k,l)\in \text{Page}_i} fp(k,l;T) \tag{9.29}$$

Similarly as for true positives, a *false positive* is defined by:

$$fp(k,l;T) = \left[\Omega_{\boldsymbol{w}}\left(\hat{\boldsymbol{y}};\mathcal{B}(\hat{h};\boldsymbol{x})\right) \geq T\right]\left[\Gamma_{k,l}(\hat{h},h^*) < 0.5\right]. \tag{9.30}$$

In other words, if the predicted sequence $\hat{\boldsymbol{y}}$ for example $\boldsymbol{x} = (\mathcal{P}_k, \mathcal{P}_l)$ scores above T, but has insufficient min-overlap entailed by its transformation $\hat{h}$, then $(\hat{h}, \hat{\boldsymbol{y}})$ counts as a false alarm. We emphasize that the only difference between a true positive and a false positive lies in the correctness of the alignment.

For an easier understanding consider the two Hinton diagrams shown in figure 9.2, which have block-diagonal form to separate pieces from different pages (8 pieces per page). Each cell represents one piece-pair. The left diagram illustrates the pieces' max-connectivity (larger patches mean higher values). Accordingly, this first diagram can

be understood as a weighted adjacency matrix. For the second diagram on the right, patches in each cell are colored according to the sign of the ssvm score (green patches for positive values, red for negative values). Using the sign of the ssvm score implies $T = 0$. The background of a cell is colored in purple if the predicted hypothesis entails a min-overlap with the ground truth of at least 0.5.

The observation from those two diagrams is that structured output prediction allows us to recover the adjacency relationship among pieces with high confidence. For each page in our example, the ssvm correctly identifies 10 spatial configurations with non-negative score and incorrectly makes a prediction for only 1 non-adjacent piece-pair[1].

Based on our definition of true positives and false positives as functions of T we now define the terms *precision* and *recall*:

$$Prec_i(T) = \frac{TP_i(T)}{TP_i(T) + FP_i(T)} \tag{9.31}$$

$$Rec_i(T) = \frac{TP_i(T)}{P_i} \tag{9.32}$$

For the computation of recall we define the quantity P_i as the number of strongly adjacent piece-pairs within page i. In contrast to binary classification, P_i can in general not be computed as the sum of true positives and false negatives. The reason for this is that finding the correct spatial configuration is an integral part of the prediction task. In fact, from eq. (9.27) and eq. (9.29) it becomes clear that

$$\widetilde{P}_i = TP_i(T) + FN_i(T) = \sum_{(k,l)\in \text{Page}_i} \left[\Gamma_{k,l}(\hat{h}, h^*) \geq 0.5\right], \tag{9.33}$$

only counts the number of adjacent piece-pairs that are aligned correctly in terms of min-overlap. In most practical scenarios we have $\widetilde{P}_i < P_i$ because the optimal transformation at test time is not known, and for some examples the min-overlap could be less than 0.5. Referring to our above example, we can only successfully recover 10 out of 12 adjacent piece-pairs for each page (a recall of 83.3%), regardless of the choice of T, showing that it can be very challenging to even achieve a recall close to 1.

[1] Note that the two empty cells correspond to *hard examples* (adjacent pieces with less than four inliers) for which there is no prediction because they are excluded from the evaluation.

9.6.2 Precision-Recall (PR) Curve

Both precision and recall can be understood as single operating points of the classifier with regards to a fixed choice of threshold T. Performing a parameter sweep of T is a principled mechanism to explore operating point tradeoffs, for which we consider the precision and the recall at every position of the ranked list. This characterizes both precision and recall as function of the ssvm score. The functional relationship between the two quantities is then plotted and the quality of the ranked list is assessed by computing the *average precision* (AP) as the area-under-curve. For our discrete rankings we determine the area-under-curve as follows:

$$AP(q) = \sum_{i=1}^{n} \big(Rec_q(T_i) - Rec_q(T_{i-1})\big) Prec_q(T_i) \tag{9.34}$$

Here n is the number of distinct ssvm scores in the ranked list of examples from the q-th page, which are given by $T_1, \ldots, T_n$ in descending order. For the border case we set $T_0 = +\infty$.

9.6.3 mean Average Precision (mAP)

The *mean Average Precision* (mAP) is computed as the average over the AP from multiple ranked lists. Our evaluation treats each page separately and considers only piece-pairs from that page to create one ranked list. Given that there are a total of M pages in the dataset we compute the mAP by:

$$mAP = \frac{1}{M} \sum_{q=1}^{M} AP(q) \tag{9.35}$$

In the following we refer to performances as AP@list and mAP@list to distinguish it from our second evaluation scenario introduced in chapter 10, where we quantify the quality of reconstructed pages in a similar manner.

9.6.4 Experiments

Next we conduct experiments to evaluate the effectiveness of our structured output prediction approach, both in terms of quality of results and computational efficiency.

Evaluation in Terms of (mean) Average Precision

For the first experiment we rank piece-pairs from the test set of the *bdw082010* dataset.

Approach	mAP@list on *bdw082010*			
	8 pieces	16 pieces	24 pieces	32 pieces
ssvm w/ MSAC ($K = 1$)	0.8849 [0.8907]	0.7693 [0.7904]	0.6338 [0.6948]	0.5622 [0.6678]
ssvm w/ MSAC ($K = 5$)	**0.9095** [0.9205]	**0.8081** [0.8496]	**0.6695** [0.7677]	**0.5801** [0.7352]
binary approach	0.8714 [0.8858]	0.7474 [0.7995]	0.6130 [0.7152]	0.5332 [0.6483]

Table 9.1: Comparison of results between structured output prediction and binary classification with hill-climbing, in terms of mAP@list (see eq. (9.35)). The best performance for each fragmentation level is marked bold. The average recall across all individual pages is reported in brackets. Experiments were conduced on the *bdw082010* test set.

We compare structured prediction to our binary approach in terms of mAP@list. For the binary approach we run algorithm `graph-init-binary`, with its best performing classification threshold, and use the graph's edge weights as a measure of confidence in the spatial configurations. The mAP@list is computed for each degree of fragmentation, treating the pages' non-adjacent piece-pairs as negative examples and strongly adjacent piece-pairs as positive examples.

The results are reported in table 9.1. The number in each column is the mAP@list. Values in brackets correspond to the average recall computed from recall values of all individual pages. As can be seen from the table, the structural approach outperforms the binary approach for each degree of fragmentation. When increasing the number of hypotheses per piece pair from $K = 1$ to $K = 5$ this becomes even more pronounced.

A second observation is that performances tend to decline as the number of pieces increases. For the most part this effect can be attributed to a lower recall rather than a drop in precision. The reason for this is that small pieces tend to have short adjacent boundary regions, which makes it inherently difficult to recover their correct spatial configuration. To illustrate that the precision still remains high throughout all experiments, let us reconsider the definition of recall previously given in eq. (9.32). Most often, a recall of 1 can not be achieved because some piece-pairs can not be aligned correctly. However, if we are merely interested in the "correctness" of our predictions and not the "completeness" of ranked lists, one can substitute the number of predictions $\widetilde{P}_i = TP_i(T) + FN_i(T)$ from eq. (9.33) for P_i. Thereby we obtain recall values which are normalized to $[0, 1]$ for any given page. In light of this alternative definition of recall, the mAP@list for the structured approach ($K = 5$) only drops from 0.9877 (8 pieces) to 0.9085 (32 pieces). We can conclude that, in spite of having fewer correctly aligned examples for higher fragmentation levels, the average precision still remains very high.

We want to emphasize that recall most often is not the limiting factor for the re-

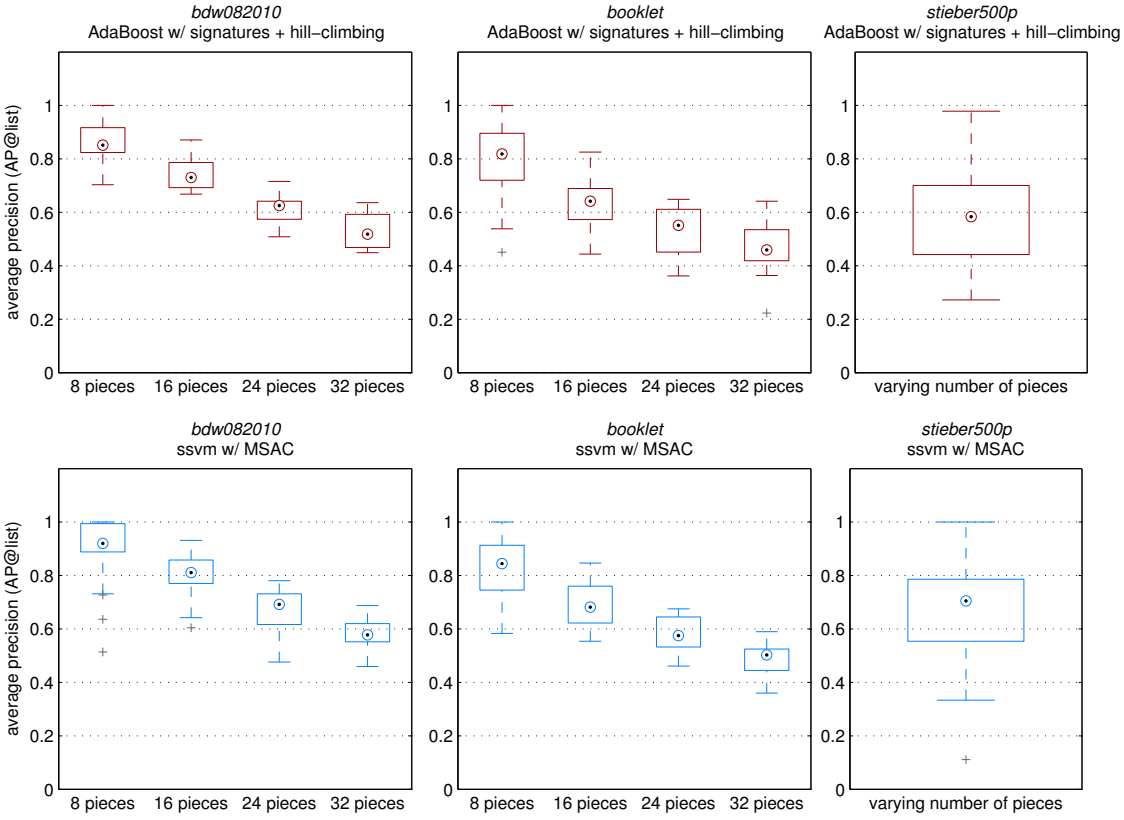


Figure 9.3: Performance statistics regarding the average precision of ranked lists, for the binary approach (top row) and the structured prediction approach (bottom row). Each column summarizes experimental results from one dataset. Box plots are used to indicate the deviation of the AP@list across all test pages. Our structured approach used the top $K = 5$ non-maximum suppressed hypotheses.

construction of pages, because hand-torn documents inherit a certain degree of redundancy regarding the adjacency of piece-pairs. That is, pieces are typically adjacent to multiple other pieces in the manually reconstructed document. Our conjecture about the importance of recall is confirmed in the next chapter, where we show that document pages with high degree of fragmentation can be reconstructed successfully despite the inherently lower recall.

A slightly different perspective on the quality of ranked lists is provided in figure 9.3. There we show performance statistics in form of box plots, one for each experiment conducted for different degrees of fragmentation. Each column summarizes experimental results from one dataset. Box plots are used to indicate the variance of AP across all pages. The circle in each box plot marks the median of the AP values

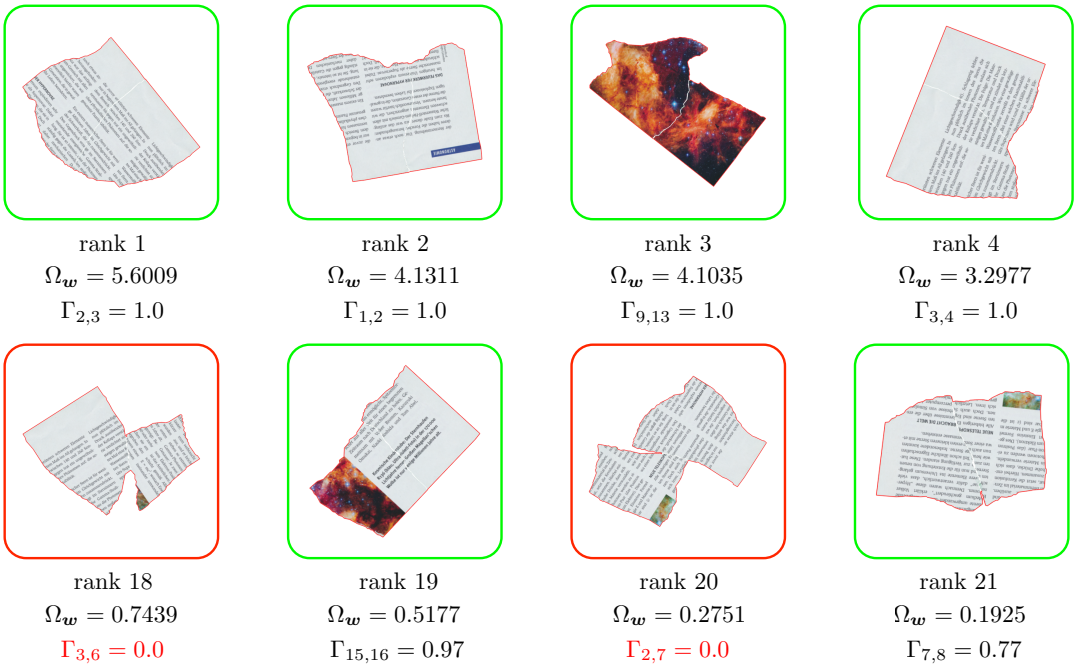


Figure 9.4: Left to right, top to bottom: Ranked list of aligned piece-pairs from a single page of the *bdw082010* dataset (16 pieces). True positives regarding threshold $T = 0$ are shown with a green border, false positives are highlighted with a red border. Note that the first false positive occurs at rank 18 in the list.

computed from all ranked lists. These plots make apparent that examples from some pages are harder to rank than others. According to our experience, pages with characteristic image content provide examples that are easier to rank correctly. Conversely, pieces from blank pages are often indistinguishable in terms of content-based features, which is the main reason why there are outliers in the box plots (crosses).

Example for Ranked Piece-Pairs

An example for a ranked list of aligned piece-pairs is given in figure 9.4. The top row shows the first four examples with the highest ssvm scores among all examples[1]. All of these pieces have been aligned correctly, as is indicated by their min-overlap values of 1.0. True positives regarding classification threshold $T = 0$ are highlighted in green, false positives in red. We note that the first false positive occurs at rank 18 in the list, as shown in the bottom row.

Utilizing ranked lists could be an invaluable asset to guide human experts in a semi-automated reconstruction procedure. Since pieces are aligned automatically, the

[1] Only the best of the $K = 5$ non-maximum suppressed hypotheses is used for each piece-pair.

user only needs to validate a small part of the list, starting from the proposed solutions ranked with highest confidence. For the examples depicted in figure 9.4, the structural SVM ranks 17 correct proposals *before* the first incorrectly aligned piece-pair. We argue that the head of this ranked list could be reviewed quickly by a human user and stored for further processing.

Computational Efficiency

Our final experiment evaluates our approach in terms of computational efficiency. We determine the average CPU time required for the *preprocessing of individual pieces* (i.e., precomputing L-neighborhoods and signed distance maps). Besides, we average the time requirements across all piece-pairs for *structured output prediction* with $K = 5$ (i.e., running MSAC and inference of the best output $f_{\boldsymbol{w}}(\boldsymbol{x}) = (\hat{h}, \hat{\boldsymbol{y}})$). For the 16 piece variant of *bdw082010* the preprocessing takes only $0.0335s$ per piece. Identifying the set of $K = 5$ non-maximum suppressed hypotheses with subsequent computation of the ssvm score requires $0.8353s$. This means that more than $100{,}000$ piece-pairs can be processed per day on a single-threaded system. Since our approach is straightforward to parallelize across multiple threads, one can easily achieve almost a linear speed-up on a multicore system.

9.7 Summary

In this chapter we have shown how piece-pairs can be ranked by means of structured output prediction, a supervised learning paradigm that is used in many applications of computer vision, including natural language processing, object detection and human pose estimation. To make it feasible in our problem setting, we have introduced a linear cost model that incorporates geometric and content-based features. Our discriminative model has been obtained through training on a set of annotated examples using a stochastic gradient descent solver. One of the key benefits of supervised learning is that it eliminates the need for a manual adjustment of thresholds in order to balance the contribution of different features. Thus, our approach is extremely flexible in that it allows an effortless integration of additional features. Finally, we thoroughly evaluated our proposed ranking method on three datasets and showed that it is both very efficient and effective. For this, we demonstrated how to adapt the (mean) average precision to our application. Using this standard performance measure commonly employed in various research domains has enabled an easily accessible and rigorous quantitative evaluation.

Chapter 10

Agglomerative Reconstruction of Multiple Pages

10.1 Introduction

This chapter discusses how to incorporate structured output prediction into our spanning tree algorithm presented in chapter 7. To this end we introduce a revised version of algorithm `kruskal-binary`. There are two major conceptual changes: First of all, because our inference algorithm unambiguously chooses a single spatial configuration for each piece-pair, the document structure no longer needs to be represented by a multigraph. As a consequence thereof, each pair of nodes is now connected by at most one edge. The second change is tied to the computation of weights for these edges. Recall that our binary approach used normalized objective function values obtained through hill-climbing. A crucial step was to re-normalize these edge weights once after pieces were merged into a larger group of aligned pieces. This repeated normalization ensured the comparability of spatial configurations, as all edge weights were constrained to the same value range. We have argued that this enables the pruning of "non-promising" edges, i.e., incorrect spatial configurations, which in turn speeds up the reconstruction.

This normalization scheme now becomes obsolete due to our maximum margin approach employed in chapter 9. Using a discriminative model always gives meaningful scores reflecting an example's distance from a decision boundary. In case of a structural SVM, this distance can be interpreted as the confidence in our prediction, based on which we discriminate correct from incorrect spatial configurations. Therefore, our revised algorithm `kruskal-struct` assigns edge weights according to the

discriminant function $\Omega_{\boldsymbol{w}}$. Since our cost model $\boldsymbol{w}$ does not change once learned, we no longer need to update edge weights to account for new normalization constants.

This chapter is organized as follows: In section 10.2 we first discuss the necessary modifications to our algorithm. Afterwards, section 10.3 introduces an extension that enables the simultaneous two-sided reconstruction of document sheets. To make this a viable option we need to identify piece-pairs stemming from the same fragment once a priori. We dub this preprocessing the "shape registration" step. Having identified the fragments' two *counterparts* enables the simultaneous merging on both sides of the document sheet. Note that two sides give us more information about the visual compatibility of two pieces along the adjacent boundary. Finally, in section 10.4 we demonstrate experimentally that our revised algorithm immediately applies to the reconstruction of multiple pages. In this very challenging scenario we neither assume to know the number of pages to be assembled nor the belonging of pieces to pages. In section 10.5 we make some final remarks on this chapter.

10.2 Graph-Based Reconstruction Algorithm

In order to use structured output prediction in our reconstruction algorithm, we first need to extend our representation of digital pieces.

10.2.1 Representation of Pieces

Instead of $\mathcal{P}_k = (S_k, \hat{S}_k, I_k)$ we now represent a piece by $\mathcal{P}_k = (S_k, \hat{S}_k, I_k, \mathcal{D}_{S_k}, \hat{\theta}_k, c(\hat{\theta}_k))$. The three added components contain all the information required for partial contour matching. More precisely, the first component $\mathcal{D}_{S_k}$ represents the signed distance map regarding the piece's outer contour S_k. The other two components are the orientation estimate $\hat{\theta}_k$ and the confidence $c(\hat{\theta}_k)$ in this estimate as obtained through our linear SVM (see section 8.6.2). Next we discuss how to deal with this new information in the following two scenarios:

(1) Applying Transformations to Pieces

Since the positions of pieces are updated repeatedly during the process of reconstruction, it is necessary to discuss how the three added properties can be updated accordingly. To this end we extend our definition of the operator $Z_k\langle\mathcal{P}_k\rangle$ from section 3.6 to all new components:

- **Distance maps:** The computation of signed distance maps requires only linear

time in the number of image pixels (see section 8.4). Thus, once the position of a piece changes, we simply recompute its distance map. More precisely, we determine $\mathcal{D}_{Z_k S_k}$ to account for the piece's repositioned contour. As usual, S_k is the $3 \times n(k)$ matrix storing contour points in homogeneous coordinates as column vectors, and Z_k is the 3×3 transformation matrix to position the piece.

- **Orientation estimates:** In general, Z_k not only involves a translation but also a *rotation*. Hence the orientation estimate $\hat{\theta}_k$, which was only computed for the piece once in advance, needs to be updated. To accomplish this, we determine the rotation angle of Z_k and use it to modify our initial estimate $\hat{\theta}_k$ accordingly. We denote this updated estimate by $Z_k\langle\hat{\theta}_k\rangle$, which is essentially the same estimate we would get from the repositioned piece $Z_k\langle\mathcal{P}_k\rangle$. Since there is no reason to believe that the confidence in our estimate should have changed, we do not need to alter $c(\hat{\theta}_k)$.

(2) Forming Groups of Aligned Pieces

The second set of changes is along the lines of the discussion in section 3.7, where we explained how to represent a group of aligned pieces by a single artificial piece. For the special case of $n = 3$ we illustrated how individual pieces indexed by $V = \{k_1, \ldots, k_n\}$ are combined into a new piece $\mathcal{P}_V$.

- **Distance maps:** To handle a new artificial piece $\mathcal{P}_V$ properly, we first compute its *joint outer contour* S_V from its individual pieces (see section 3.7). This allows us to compute the signed distance map with regards to the new outer contour, which is represented by $\mathcal{D}_{S_V}$.
- **Orientation estimates:** The orientation estimate $\hat{\theta}_V$ for a group of pieces can be obtained without recomputation. Since we keep track of the updated estimates $Z_k\langle\hat{\theta}_k\rangle$ for each individual piece $k \in V$, we can simply choose the estimate with the highest confidence.

Summary

Note that neither of these two steps requires a recomputation of orientation estimates from scratch. Therefore, the computation and classification of orientation estimates as discussed in section 8.6 can be treated as a mere preprocessing step. The only thing that actually needs to be recomputed is the signed distance map. However, this is a very efficient operation which amounts to negligible computational overhead.

Algorithm 10.1: Document graph initialization (`graph-init-struct`)

Input : Pieces $\mathcal{P}_1, \ldots, \mathcal{P}_N$
Parameters : Cost model $\boldsymbol{w}$
Output : Initialized document graph $\mathcal{G}$

Initialization
  Declare $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E})$ with $\mathcal{V} = \{1, \ldots, N\}$, $\mathcal{E} = \emptyset$, and
  $l_\mathcal{V} = (\Sigma_Z) = (\emptyset)$
  $l_\mathcal{E} = (\Sigma_H, \Sigma_Y, \Sigma_W) = (\emptyset, \emptyset, \emptyset)$

Set up nodes
  `/* meta-information for nodes */`
  foreach $k = 1, \ldots, N$ **do**
    `/* set transformation for positioning randomly */`
    $\Sigma_Z[k] = T_{rand,k}$

Set up edges
  `/* consider all (unique) piece-pairs */`
  for $k < l$ **do**
    `/* best prediction regarding cost model */`
    $(\hat{h}, \hat{\boldsymbol{y}}) = f_{\boldsymbol{w}}(\boldsymbol{x})$, with $\boldsymbol{x} = (\Sigma_Z[k]\langle\mathcal{P}_k\rangle, \Sigma_Z[l]\langle\mathcal{P}_l\rangle)$
    if $\hat{\boldsymbol{y}} \neq \emptyset$ **then**
      `/* add edge to edge set */`
      $\mathcal{E} = \mathcal{E} \cup \{e\}$, with $e = (k, l)$
      `/* meta-information for edges */`
      $\Sigma_H[e] = \hat{h}$, $\Sigma_Y[e] = \hat{\boldsymbol{y}}$, $\Sigma_W[e] = \Omega_{\boldsymbol{w}}(\hat{\boldsymbol{y}}, \mathcal{B}(\hat{h}; \boldsymbol{x}))$

return $\mathcal{G}$

10.2.2 Graph Initialization

The initialization procedure to obtain a document graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E})$ through structured output prediction is almost the same as the one put forward in chapter 7. The most important difference is that, instead of using hill-climbing, we apply MSAC (see chapter 8) in conjunction with our structural SVM (see chapter 9) to identify the presumably best spatial configurations. Since we choose only one spatial configuration per piece-pair, the document graph for this problem formulation is no longer a multigraph.

The revised version of algorithm `graph-init-binary`, which was based on binary classification, is presented in `graph-init-struct`. Note that the use of struc-

tured output prediction made it conceptually a lot more straightforward: First of all, we require less meta-information to be associated with the edges. In $l_{\mathcal{E}} = (\Sigma_H, \Sigma_Y, \Sigma_W)$ we now only store the structured outputs (rigid transformations and sequences), as well as the associated ssvm scores, which are used as edge weights. We note that most often when no valid spatial configuration between pieces can be found, this is because their adjacent boundary regions have insufficient length. That is, in this degenerated case of adjacent boundary regions consisting of only a single support point, we set the sequence $\hat{\boldsymbol{y}} = \emptyset$. In this case we create no edge between the two nodes (see line 9). Also note that in contrast to our binary approach there is no need for a re-normalization of edge weights.

10.2.3 Agglomerative Reconstruction

In comparison with its binary equivalent (see algorithm `kruskal-binary`), our structural approach `kruskal-struct` requires only a few changes.

First of all, instead of employing a pruning heuristic like in the binary approach, we now impose an explicit threshold on the edge weights (line 8). Thereby we disregard piece-pairs with insufficient compatibility and thus avoid to create partial solutions that are most likely incorrect. Our algorithm terminates in this case, as no further progress on reconstruction can be made after this point.

A second less subtle difference lies in the sub-procedure `update-graph-struct`. We update the graph's link structure in three steps: First, we remove any edge connecting pieces from the new cluster V with any of the other clusters (line 4). We then solve the joint inference task with the dynamic programming method discussed in section 9.5.4 (line 8). Thereby we consider the newly merged piece $\mathcal{P}_V$ to account for the additional evidence of the last merging step. In the last step we choose the two pieces $k \in V, l \in V'$ that share the longest adjacent boundary after alignment (line 10). We want to emphasize that this is only required for our quantitative evaluation and has no impact on the actual reconstruction result.

10.3 Simultaneous Two-Sided Reconstruction

Although our algorithm has been devised for a one-sided reconstruction of pages, it can easily be changed into a duplex version. That is, instead of considering individual pages on their own, we perform a *simultaneous* reconstruction on both sides of a given document sheet. The benefit of a two-sided reconstruction is that within each iteration two piece-pairs instead of a single one can be merged. Intuitively, even if we

Algorithm 10.2: Spanning tree algorithm (`kruskal-struct`)

Input : Pieces $\mathcal{P}_1, \ldots, \mathcal{P}_N$
Parameters : Cost model $\boldsymbol{w}$
Output : Set of transformations $\{Z_1, \ldots, Z_N\}$, spanning tree edges $\mathcal{E}_{span}$

Initialization
- $\mathcal{G} = (\mathcal{V}, \mathcal{E}, l_\mathcal{V}, l_\mathcal{E}) \leftarrow$ `graph-init-struct`$(\mathcal{P}_1, \ldots, \mathcal{P}_N; \boldsymbol{w})$
- $\mathcal{E}_{span} = \emptyset$
- `/* set up clusters */`
- $C = \{V_1, \ldots, V_N\}$ with $V_k = \{k\}$

Agglomerative reconstruction
- **while** $|C| > 1$ **do**
  - `/* choose best transformation (if any) */`
  - $e^* = (k^*, l^*) = \underset{\substack{e=(k,l)\in\mathcal{E} \\ idx(k)\neq idx(l)}}{\operatorname{argmax}} \Sigma_W[e]$
  - `/* stop if no (valid) transformation was found */`
  - **if** $e^* = \emptyset$ *or score insufficient with* $\Sigma_W[e^*] < T_{ssvm}$ **then**
    - break
  - `/* merge nodes into a single cluster */`
  - $V = V_{idx(k^*)} \cup V_{idx(l^*)}$
  - `/* update the pieces' positions */`
  - **foreach** $l \in V_{idx(l^*)}$ **do**
    - $\Sigma_Z[l] = \Sigma_H[e^*]\Sigma_Z[l]$
  - `/* update the set of clusters */`
  - $C = C \setminus \{V_{idx(k^*)}, V_{idx(l^*)}\} \cup \{V\}$
  - `/* add edge to spanning tree */`
  - $\mathcal{E}_{span} = \mathcal{E}_{span} \cup \{(k^*, l^*)\}$
  - `/* update edge labels regarding new cluster */`
  - $\mathcal{G} =$ `update-graph-struct`$(\mathcal{G}, V, C \setminus V; \boldsymbol{w})$

return $\{Z_k = \Sigma_Z[k] \mid k = 1, \ldots, N\}, \mathcal{E}_{span}$

encounter a blank document page, the back side of the sheet may show content that mitigates the lack of information on the front side. Considering both sides of the same fragment, to which we refer as *counterparts*, can be seen as using an additional source of information. Thus, we conjecture that making simultaneous merges leads to better reconstruction results.

Algorithm 10.3: Update graph (`update-graph-struct`)

Input : Document graph $\mathcal{G}$, new cluster V, other clusters $C \backslash V$
Parameters : Cost model $\boldsymbol{w}$
Output : Updated graph

Update edge structure and labels

`/* remove old edges associated with new cluster */`

foreach $e = (k, l) \in \mathcal{E}$ **do**

  if $V_{idx(k)} = V \; XOR \; V_{idx(l)} = V$ **then**

    $\mathcal{E} = \mathcal{E} \backslash \{e\}$

`/* re-configure old clusters with new cluster */`

foreach $V' \in C \backslash V$ **do**

  `/* treat clusters as artificial new pieces */`

  $\mathcal{P}_V \longleftarrow$ `group`$(\{\Sigma_Z[k] \langle \mathcal{P}_k \rangle \mid k \in V\})$

  $\mathcal{P}_{V'} \longleftarrow$ `group`$(\{\Sigma_Z[l] \langle \mathcal{P}_l \rangle \mid l \in V'\})$

  `/* best prediction regarding cost model */`

  $(\hat{h}, \hat{\boldsymbol{y}}) = f_{\boldsymbol{w}}(\boldsymbol{x})$, with $\boldsymbol{x} = (\mathcal{P}_V, \mathcal{P}_{V'})$

  if $\hat{\boldsymbol{y}} \neq \emptyset$ **then**

    `/* choose inter-cluster piece-pair with longest adjacent boundary */`

    $\hat{e} = (k, l) = \text{argmax}_{k \in V, l \in V'} \{l(B_k(\hat{h})) + l(B_l(\hat{h}))\}$

    `/* add the new edge */`

    $\mathcal{E} = \mathcal{E} \cup \{\hat{e}\}$

    `/* meta-information for new edge */`

    $\Sigma_H[\hat{e}] = \hat{h}$, $\Sigma_Y[\hat{e}] = \hat{\boldsymbol{y}}$, $\Sigma_W[\hat{e}] = \Omega_{\boldsymbol{w}}\big(\hat{\boldsymbol{y}}, \mathcal{B}(\hat{h}; \boldsymbol{x})\big)$

return $\mathcal{G}$

10.3.1 Identifying Counterparts through Shape Registration

Performing two-sided reconstruction requires an additional preprocessing step that we refer to as *shape registration*. The necessity of identifying counterparts arises from using an off-the-shelf scanner without duplex scanning functionality: Different sides of the same fragments were generally not scanned in the same order. Thus, the correspondence of each piece to its counterpart is not known in advance. Moreover, pieces from different sides of the same document sheet have been digitized individually under arbitrary rigid transformations. Correspondingly, even if we were to know a fragment's two counterparts, we still do not know the exact transformation to register one

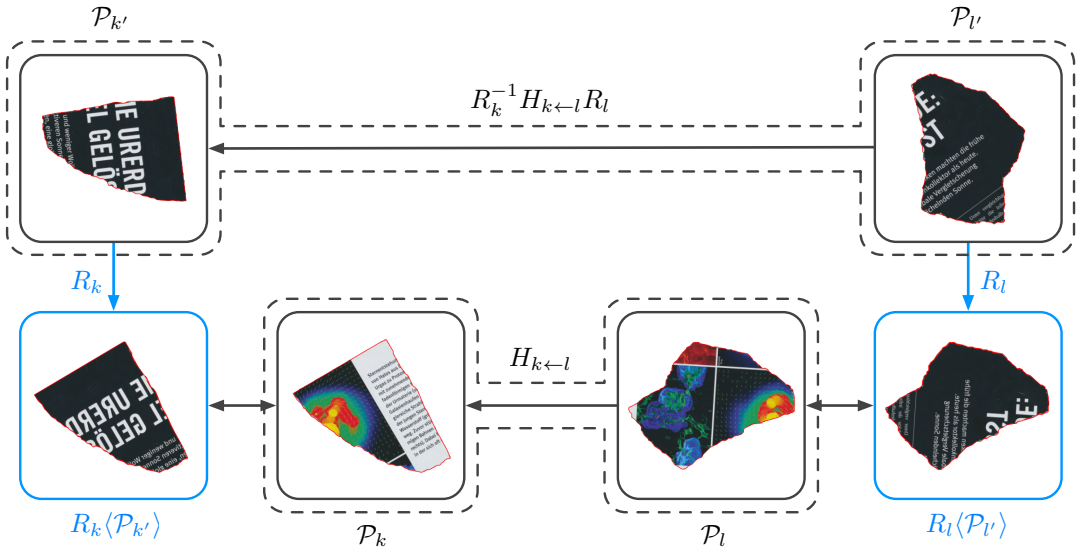


Figure 10.1: Example for two piece-pairs that can be registered as counterparts. For instance, applying transformation R_l to $\mathcal{P}_{l'}$ yields a mirrored version of that piece, which is superimposed with its counterpart $\mathcal{P}_l$. To register two pieces, a reflection around the x-axis must be applied prior to running MSAC. See text for details.

piece with the other.

We discover counterparts automatically by considering small chunks of pieces that have been digitized in immediate succession. This reflects our digitization methodology with which we scanned fragments of one sheet from one side before proceeding likewise with their back side. Apart from minor discrepancies in the contours due to noise, two counterparts are precise *congruent* versions of one another. It is clear that those pieces have the same size and differ only in that their outer contours are exactly mirrored.

Using our partial contour matching one can recover counterparts by finding transformations that superimpose pieces appropriately. To account for the fact that contours are mirrored, we first reflect one of the pieces before computing the actual transformation. An example for successfully discovered counterparts is given in figure 10.1. Here, both piece-pairs $(\mathcal{P}_k, \mathcal{P}_{k'})$ and $(\mathcal{P}_l, \mathcal{P}_{l'})$ form counterparts which are registered through transformations R_k and R_l, respectively.

10.3.2 Modifications for Two-Sided Merging

From an algorithmic perspective, performing a two-sided instead of a one-sided reconstruction is relatively straightforward. Once the two most compatible clusters have

been chosen, we start by merging them as in the one-sided case. With regards to the example in figure 10.1, we would first merge $\mathcal{P}_k$ and $\mathcal{P}_l$ by repositioning the latter through transformation $H_{k \leftarrow l}$. Afterwards, we "propagate" $H_{k \leftarrow l}$ to the sheet's other side by applying the transformation to the pieces' counterparts. However, to perform this second merge, we also need to incorporate the rigid transformations for shape registration properly. In our example, we align piece $\mathcal{P}_{l'}$ with $\mathcal{P}_{k'}$ by applying a sequence of transformations $R_k^{-1} H_{k \leftarrow l} R_l$ to the counterpart $\mathcal{P}_{l'}$.

Note that in direct comparison with the one-sided case, a two-sided variant of our reconstruction algorithm requires only half the number of iterations. The reason for this is that during each step, twice as many pieces are combined into clusters. In the following section we give a quantitative comparison of results regarding the one-sided and the two-sided reconstruction.

10.4 Evaluation

In the previous chapter we have evaluated the quality of ranked lists containing piece-pairs from individual pages. This evaluation served two purposes: To find out (i) how many piece-pairs can be aligned correctly (recall), and (ii) how reliably the ssvm score distinguishes correctly aligned piece-pairs from those that were misaligned (precision). For each ranked list we computed the average precision (AP) and finally obtained the mAP@list as the mean of all AP values.

Examples in these comprehensive lists of piece-pairs provide redundant information about the adjacency relationship within a page. Hence, in analogy with our previous evaluation in section 7.3, we argue that a subset of these piece-pairs is sufficient for recovering the document layout. Given a page with N pieces, the $N - 1$ edges of the spanning tree of a document graph identify a minimal number of adjustments to the pieces' positions. The associated $N - 1$ merging decisions can be interpreted as a short list of ranked examples that allows us to use the mAP also for the evaluation of reconstructed pages (based on their spanning trees).

10.4.1 mAP for Reconstruction of Individual Pages

Reconstruction of a document page with N pieces requires exactly $N-1$ merging decisions. Each decision is represented by one edge in the spanning tree $\mathcal{E}_{span}$ that results from running algorithm `kruskal-struct`. Similarly as for the comprehensive lists of ranked piece-pairs in the previous chapter, each decision is either a true positive or a false positive.

We evaluate the number of true positives in the list of spanning tree edges by:

$$TP_i(T) = \sum_{(k,l)\in\mathcal{E}_{span}} tp(k,l;T) \tag{10.1}$$

Note that this definition differs from the one for comprehensive lists (eq. (9.27)) in that it considers only spanning tree edges instead of a *full list* of piece-pairs. Each edge $e = (k,l) \in \mathcal{E}_{span}$ now has the intuitive meaning of connecting two clusters (groups of pieces) instead of only two individual pieces. Only during graph construction clusters consist of individual nodes and thus edges still have the meaning of aligning piece-pairs. In general, when considering two clusters V and V' for merging, we have to choose a representative edge between them to define the spanning tree unambiguously. We thus choose the piece-pair $(k,l) \in V \times V'$ for which the adjacent boundary regions have maximal length (line 10 in algorithm `update-graph-struct`).

We assess whether those pieces are aligned correctly by:

$$tp(k,l;T) = \left[\Omega_{\boldsymbol{w}}\left(\hat{\boldsymbol{y}};\mathcal{B}(\hat{h};\boldsymbol{x})\right) \geq T\right]\left[\Gamma_{k,l}(\hat{h},h^*) \geq 0.5\right]. \tag{10.2}$$

Here each example consists of two artificial pieces $\boldsymbol{x} = (\mathcal{P}_V, \mathcal{P}_{V'})$. For the evaluation of false positives one can proceed analogously.

The last modification relates to the computation of recall. In contrast to comprehensive lists for which we set P_i to the number of strongly adjacent piece-pairs (see eq. (9.32)), we now define $P_i = N - 1$. Since this is the number of edges in the spanning tree, we obtain a recall of 1 only if all predicted spatial configurations are correct, which indicates a complete reconstruction of the given page. Furthermore, predictions regarding hard examples must not be ignored in the reconstruction scenario because P_i is a fixed number that does not rely on the inlier counts; if these predictions were omitted, a recall of 1 could not be obtained if at least one piece is connected to all other adjacent pieces by less than 4 inliers.

In analogy with AP@list and mAP@list we refer to the performance measures for the evaluation of reconstruction results by AP@span and mAP@span, respectively.

10.4.2 Experiments

We now report on extensive experiments to evaluate our revised algorithm.

Reconstruction of Individual Pages

Our first series of experiments is very similar in spirit to those conducted in the last

chapter. The difference is that, instead of reporting performances for comprehensive ranked lists in terms of AP@list, we reconstruct pages individually and evaluate the quality of the reconstruction results in terms of AP@span.

The results are summarized in figure 10.2. Plots in the first row regard our binary approach. Each row has three groups of box plots, one for each dataset. Each individual box plot reports performances in terms of AP@span for one degree of fragmentation. The last two rows give results obtained by performing one-sided and two-sided reconstruction, respectively.

Our first observation is that the one-sided structural approach performs seemingly worse than the binary approach. We have found that this is not a matter of precision, but rather is due to a lack of recall. There are two main reasons for this effect: First of all, if an incorrect orientation estimate is assigned to a piece, this always leads to incorrect hypotheses from MSAC, provided that this estimate is used in the first rejection step (see section 8.5.1). If this is the case, the piece's hypotheses are likely to be rejected by the ssvm (due to an insufficient score), meaning that it will be left out from the reconstruction. The second reason is that even some of the correct spatial configurations may not obtain a sufficiently high ssvm score (above threshold $T_{ssvm} = 0$).

In any of these two cases, the algorithm stops early (line 8) and returns a set $\mathcal{E}_{span}$ that does not form a spanning tree, but rather a spanning forest consisting of *multiple* spanning trees. Each of these trees identifies one connected component of the document graph and thus represents a partial solution to the problem. For instance, if two edges were missing, $\mathcal{E}_{span}$ forms three spanning trees. Accordingly, the page is reconstructed only partly in that it consists of three separate partial solutions that can not be assembled any further.

As can be seen from the plots in the bottom row, switching from a one-sided to a two-sided merging strategy significantly boosts the performance. Our algorithm is now capable of mitigating the lack of information on one side of the sheet: As illustrated before in figure 10.1, making merges for both counterparts in the same iteration requires only one hypothesis for either of the two pieces (and only one correct orientation estimate). Consequently, we can now align pieces that could not be positioned before, resulting in higher recall values and better performances in terms of AP@span. To a certain extent, one could also try to increase the recall by lowering threshold T_{ssvm}, which was set empirically to 0. However, we want to emphasize that this would also lead to more false positives, i.e., a drop in precision. We did not investigate this matter more closely, because the actual perceived quality of reconstruction results depends more on precision than on recall. One can argue that for human experts it will be less

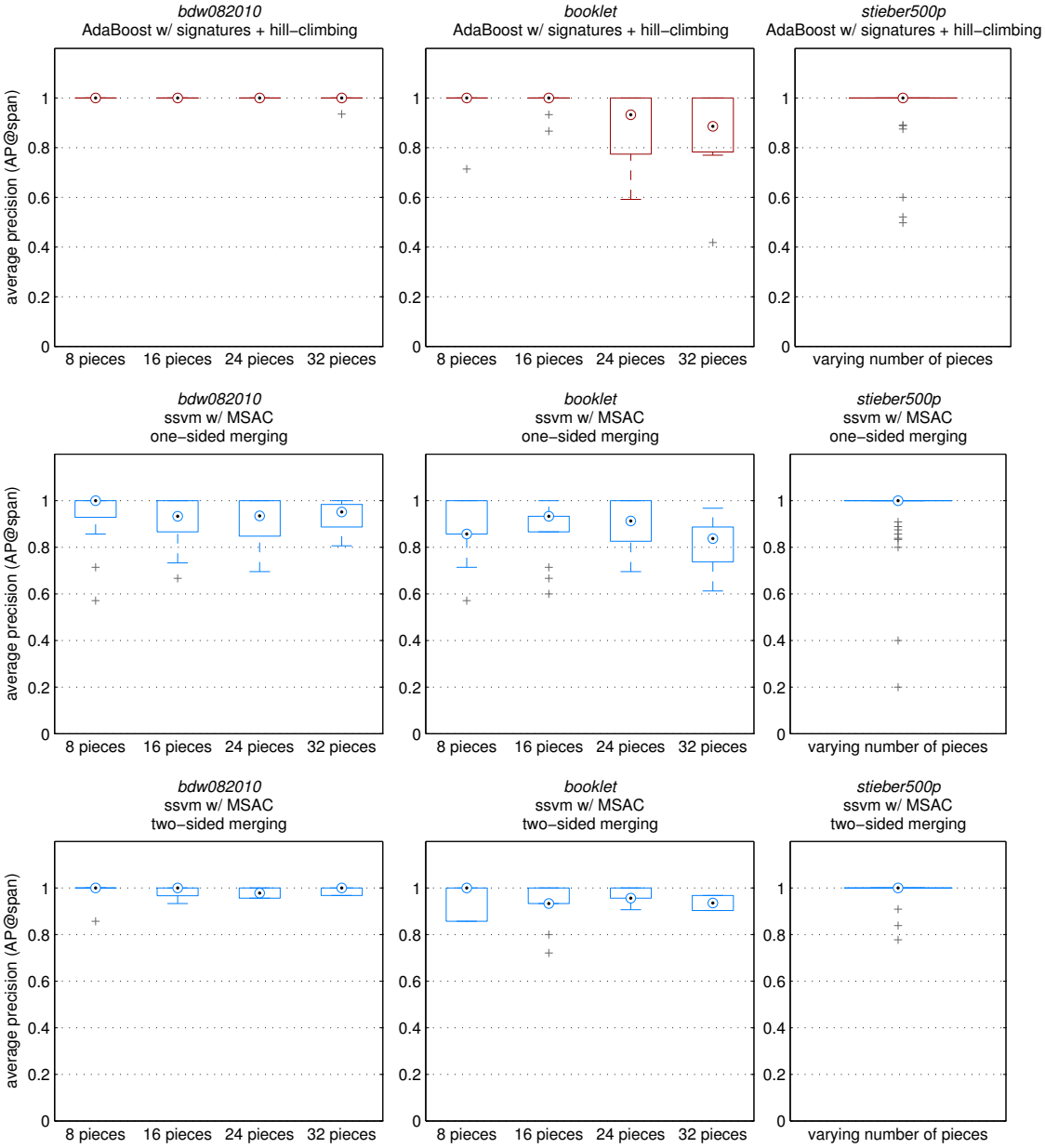


Figure 10.2: One box plot for each dataset, showing statistics of AP@span across all reassembled test pages. We used both one-sided (second row) and two-sided merging (third row) with $K = 5$ non-maximum suppressed hypotheses. For comparison we also report performances for the binary approach (first row).

tedious to combine a few correct partial solutions into a reconstructed document page, as opposed to having to identify misplaced pieces in the first place.

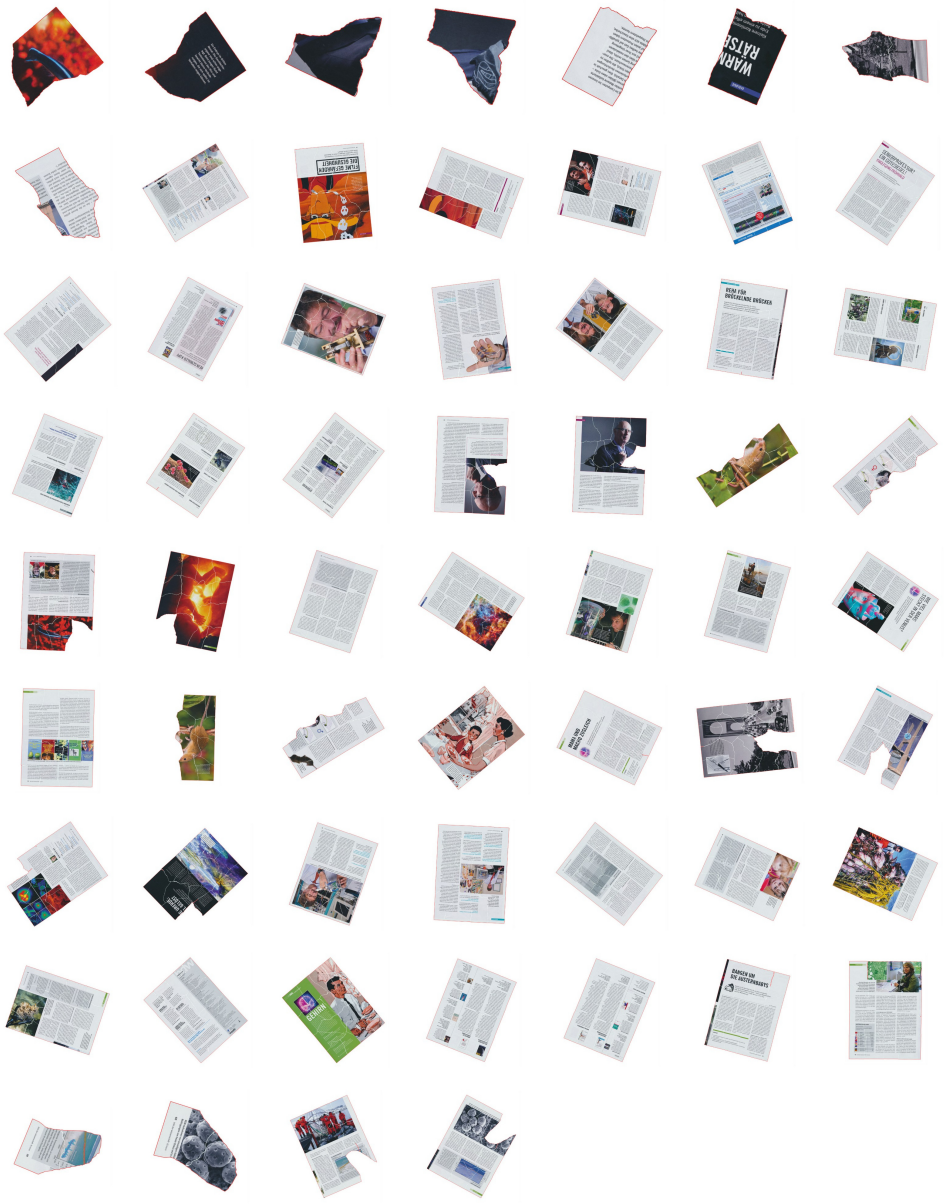


Figure 10.3: Partial solutions obtained for the *bdw082010* dataset (48 pages, 16 pieces). We used our two-sided structural approach for the reconstruction ($K = 5$ non-maximum suppressed hypotheses). Notably, none of the merging decisions was incorrect.

To get an impression of how our reassembled document pages look, consider figure 10.3. For this experiment we used the 16 piece version of the *bdw082010* dataset and two-sided merging. Overall, six out of 24 document sheets were reconstructed incompletely, with one missing spanning tree edge for each of these twelve pages. More

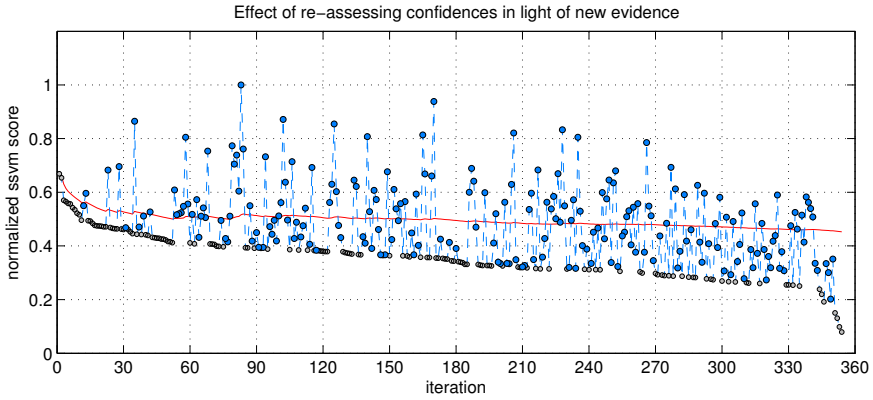


Figure 10.4: This plot shows how confidences (i.e., ssvm scores) change in light of new evidence. Since the spatial configurations and ssvm scores are recomputed once after each merging step, we obtain many upwards outliers (blue dots). An outlier indicates that the reliability of a decision increased due to a preceding merging step.

importantly, however, we made no incorrect merges, i.e., the number of false positives across all pages is 0. The fact that almost all pages have been reconstructed completely is also reflected in our mAP@span, which for this experiment is 0.9833.

Impact of Additional Evidence on Confidences

In our second experiment we analyze the impact of re-assessing the ssvm scores once after two piece-groups are merged. To this end, we perform a two-sided reconstruction of all 48 pages in the *bdw082010* test set (16 piece variant) and plot the ssvm scores against the iteration in which the pieces were merged. We show the resulting plot in figure 10.4.

Since each of the 24 sheets requires 15 merging steps, values on the x-axis range from 1 to 360. On the y-axis we plot the ssvm score, which is normalized to $[0, 1]$[1]. A blue dot indicates that due to a preceding merging step and the realignment of pieces thereafter, the current decision was made with higher confidence.

It is noteworthy that we can improve the ssvm score in 211 out of 354 iterations when taking into account the newly merged piece-group. This improvement of scores is a very strong indicator that updating the graph leads to more robust reconstruction results. The red line corresponds to the mean of all ssvm scores up to a given iteration. Since this value remains almost at a constant level, we conclude that (on average) the confidence in our decisions remains high throughout the entire reconstruction process.

[1] We use normalized values for this plot because absolute differences are less informative for the characterization of score improvements.

AP@span	*bdw082010*				*booklet*				*stieber500p*
pieces per page	8	16	24	32	8	16	24	32	varies ($\varnothing = 8.33$)
ssvm K=5	0.9941	0.9833	0.9783	0.9893	0.9405	0.9416	0.9629	0.8459	0.9863

Table 10.1: Average precision for reconstructed documents without knowing the belonging of pieces to pages.

Simultaneous Reconstruction of Multiple Pages

For our next experiment we give up our knowledge from the ground truth about the belonging of pieces to pages. This change has a very notable impact on the problem size: For illustration, let us consider the 8 piece version of the *bdw082010* test set, for which we compare two reconstruction settings. The first one is dubbed the *restricted* setting because we assume that the pieces of a given page are given. Thus, each page can be reconstructed individually. We refer to our second approach as the *unrestricted* setting because we consider all pieces simultaneously throughout the reconstruction. To make the difference between the two settings quantitative, we analyze the number of individual piece-pairs that need to be compared during graph construction:

- **Restricted (individual reconstruction of pages):** There are $M = 48$ pages that are to be reconstructed separately with $N = 8$ pieces per page. Since each piece has to be compared with every other piece exactly once, the overall number of structured output predictions for the graph construction is $M(\frac{N^2-N}{2}) = 1{,}344$.
- **Unrestricted (simultaneous reconstruction of multiple pages):** Instead of reassembling $M = 48$ pages with $N = 8$ pieces each, we perform reconstruction with $N' = 8M = 384$ pieces at the same time. During the graph construction this amounts to $\frac{(MN)^2-MN}{2} = 73{,}536$ comparisons of piece-pairs.

From a qualitative perspective, the results are almost identical in both reconstruction settings. Remarkably, for all degrees of fragmentation of the *bdw082010* test set, the results are completely identical. Although there are a few subtle differences for the *booklet* and *stieber500p* dataset, we want to point out that almost none of the additional cross-page examples were selected for merging (these reconstruction results are provided as supplementary material). One has to bear in mind that for our previous example there are $72{,}192$ cross-page piece-pairs, as opposed to only $1{,}344$ intra-page examples. It should also be noted that neither of the latter two datasets has contributed any examples to our training set, which clearly shows that our supervised learning approach is very well capable of dealing with unknown types of pieces.

For a quantitative evaluation we proceed similarly as for mAP@span. However,

the difference is that we have to account for predictions stemming from cross-page examples. Since we gave up on the prerequisite of knowing the belonging of pieces to pages, we can no longer compute a mean average precision. Instead, we compute the AP@span for which all predictions are treated jointly in one list of ranked examples. The result of two-sided merging on each dataset is reported in table 10.1.

We want to illustrate the meaning of these results and draw a comparison to related work. In [51] the authors of the *stieber500p* dataset report a reconstruction rate of 97.73% with 215 correct accepts out of 220 required two-sided merging decisions. In comparison, our two-sided structural approach (with $K = 5$) makes 221 merging decisions, of which 217 are correct. The four false positives coincide with low confidences (ssvm score close to 0) and are ranked among the last six elements in the list. Accordingly, the precision remains 1 up to a recall of 97.73%. Overall, we accomplish a recall (i.e., reconstruction rate) of 98.64%, at a AP@span of 0.9863.

Computational Effort

In the last experiment we evaluate how our method's runtime is affected by the number of pieces that need to be reassembled. To make this result meaningful for a practical application, we consider the unrestricted reconstruction setting in which pieces are not assigned to pages.

We consider two types of measurements:

- **CPU-time:** We take the elapsed time for structured output prediction as an indicator for the inherent computational effort of reassembling multiple pages. To evaluate the functional dependency between computational effort and number of pieces, we perform a two-sided reconstruction for a varying number of pages of the *bdw082010* dataset. We used between 2 and 32 randomly sampled pages, in steps of powers of 2, with 8 pieces per page. Each experiment was repeated five times for different sets of pages, and the resulting CPU-times were averaged across all experiments.

  The timing results are shown in figure 10.5. The plot draws a baseline (gray line) which we obtained through extrapolation: We took the average of the CPU-time elapsed during structured output prediction across all piece-pairs (for all 48 test pages). This is denoted by W (in seconds). As discussed in the appendix (see remark 1), the entire bottom-up reconstruction requires at most $N^2 - 4N + 4$ comparisons of piece-pairs, where N is the number of pieces. Thus, an estimate for the expected CPU-time for a complete reconstruction can be computed as $W(N^2 - 4N + 4)$. As can be seen from the figure, this estimate provides a tight

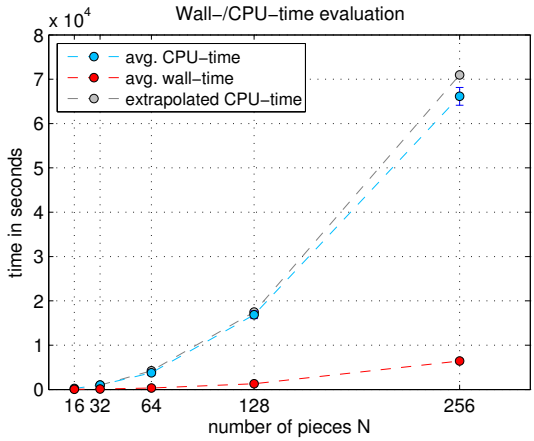


Figure 10.5: Wall- and CPU-time requirements for an unrestricted reconstruction setting, depending on the number of pages provided as input to algorithm `kruskal-struct`. We considered $M = 2, 4, 8, 16, 32$ pages with 8 pieces per page and used two-sided reconstruction. Experiments were run on a workstation with two E5540 Intel Xeon CPUs @2.53GHz (16 threads).

upper bound on the CPU-time spent, which substantiates our theoretical finding that the computational effort grows quadratically in N.

- **Wall-time:** Our second measurement is the wall-time elapsed from the start to the end of the entire reconstruction process. It includes all aspects such as feature extraction, I/O operations and the a priori shape registration for the identification of counterparts. The results are also plotted in figure 10.5. Since our approach can be parallelized with almost no data concurrency, the perceived elapsed time is considerably lower in comparison to the CPU-time. Experiments were run on a workstation with two E5540 Intel Xeon CPUs @2.53GHz, allowing parallelization across 16 threads. We believe that a well thought out implementation could easily speed-up runtime almost linearly with the number of threads.

10.5 Summary

In this final chapter we have extended our previous reconstruction algorithm to utilize structured output prediction. We have also demonstrated how a two-sided merging can be implemented. For this purpose we discussed an a priori shape registration step, which automatically identifies the two pieces stemming from the same fragment. We have demonstrated that this can be accomplished almost effortlessly using a slightly modified variant of the previously introduced MSAC approach. In our experiments

we have shown that a two-sided approach leads to essentially perfect reconstruction results, in time quadratic in the number of pieces to be reassembled.

Part IV

Conclusion

Chapter 11

Conclusion

11.1 Summary

In this thesis we have presented two approaches for the virtual reconstruction of hand-torn documents that both made use of supervised learning. Due to the lack of publicly available benchmark datasets we developed two novel, human-annotated real-world datasets. Having a ground truth of manually reconstructed documents not only enables a quantitative evaluation in different reconstruction scenarios, it also provides labeled training examples. To compensate the lack of a public benchmark, we propose two novel evaluation methods that provide meaningful insights and allow a rigorous quantitative comparison of results. Beside that, we complement our own datasets by annotating a third-party dataset, for which we report state-of-the-art results using a structured prediction approach. Our main contribution lies in the development of our two reconstruction approaches:

Approach 1: Reconstruction based on Binary Classification
Our first approach employs adaptive boosting for the classification of contour points. The purpose of this classifier was to identify points across two pieces that were putatively adjacent in the original document. The classifier relies on a description of point-pairs that encodes pairwise dissimilarities regarding a diverse set of geometric and content-based features. We showed experimentally that a large part of incorrect point-pairs can be discarded while retaining the majority of correct examples. Positive predictions obtained from the classifier form the basis for the recovery of the pieces' optimal spatial configuration. Consequently, the effort for aligning pieces heavily depends on the number of predictions.

To further reduce the number of incorrect predictions, we decided to postprocess the classification results using a spatial verification through geometric signatures. In essence, each signature puts two positive predictions on one piece into a geometric context with each other by encoding the points' relative spatial arrangement. We then utilize a voting procedure that compares signatures extracted from different pieces. As shown in our experiments this strategy proves to be very effective for invalidating misclassifications. Our voting procedure reliably identifies the majority of erroneous predictions, which narrows down the number of spatial configurations to a very manageable size.

To solve the actual reconstruction task, we create a document graph which represents a multitude of different spatial configurations of piece-pairs. The link structure of this graph is defined based on the pieces' verified point-pairs. Then, our algorithm iteratively chooses the most compatible pair of pieces according to the output of a hill-climbing optimization. A key aspect of this procedure is that the graph's link structure and edge weights are updated once after each merging step. Hence it allows us to incorporate additional evidence gathered throughout the reconstruction, for instance, in form of longer outer contours of piece-groups that were merged previously. Finally, we conducted experiments on all three datasets using a novel performance measure dubbed the mean Adjustment Cost (mAC). Since the mAC assess the quality of results based on the ground truth, it enables an objective and rigorous evaluation.

In spite of the fact that our results were very accurate, we felt that there was still room for improvement regarding the runtime of the proposed method.

Approach 2: Reconstruction based on Structured Output Prediction

The main focus of our second approach was on computational efficiency because this is a prerequisite for dealing with large-scale real-world problems. The most important insight was that classification can be postponed to the point where we need to verify the compatibility of pieces, i.e., *after* their alignment. To account for this conceptual change, we proposed a novel variant of the M-estimator SAmple Consensus (MSAC) method. Since our approach exploits information from the pieces' polygonal curves, it is tailored specifically to the alignment of piece-pairs but can be implemented very efficiently. We argued that this makes it much more efficient in comparison with a standard implementation of the closely related RANdom SAmple Consensus (RANSAC) method.

To solve the actual reconstruction task, our second approach utilizes a different supervised learning paradigm – instead of binary classification, we employed struc-

tured output prediction because it enables a sound theoretical problem formulation. We presented a revised graph-based algorithm which takes advantage of a structural support vector machine. To compare our two approaches we used the average precision (AP), which is widely recognized as the standard evaluation technique in object detection and image retrieval systems. Finally, we thoroughly evaluated the structural approach on all three datasets and showed that it outperforms our binary approach in different problem settings. Even in an unrestricted reconstruction scenario where not even the number of documents is known, the structural approach achieves excellent reconstruction results.

11.2 Conclusion and Outlook

In conclusion, it should be emphasized that this thesis does not claim to present an approach that ultimately solves the problem of reconstructing hand-torn documents. Clearly, a treatise on the most general form of this problem would have been beyond the scope of this work. As we have concentrated on dealing with this problem from a scientific perspective, we may have neglected some of the issues one has to deal with in practice.

First of all, due to the lack of publicly available benchmarks, we had to resort to create our own datasets. Many scientific datasets, with ours being no exception, have some sort of bias. For example, we have torn document sheets individually instead of stacking them, and we assumed that none of the fragments has been lost. However, we do believe that using our algorithm for the reconstruction of incomplete pages will still give very satisfactory results. The only key difference between our methodology and one considering missing pieces is that in the latter scenario, document pages offer less redundant possibilities for the relative positioning of pieces. That is, if two pieces stem from "adjacent positions" in the original document, but one of them was missing, then the other one has effectively one less correct matching partner. If many pieces are missing from a dataset, this inevitably incurs a notable loss of information. However, because we are able to reliably assess the compatibility of piece-pairs and accept only those with sufficiently high score, this lack of redundancy in matching partners would arguably lead to a drop in recall (completeness of reconstruction) rather than precision (correctness of results). Moreover, our proposed evaluation scheme would need only minor modifications to deal with missing pieces properly. Thus, the entire framework could be easily adapted to meet these new requirements. In spite of the fact that there are some limitations, we do think that this work provides an excellent starting point

for further investigation of the problem.

Another promising research direction for future work could be the analysis of different paper types. That would make it possible to partition the document graph into several smaller block matrices, thereby greatly reducing the number of pairwise comparisons of pieces. This would provide a significant speed-up to our method, making it applicable to substantially larger problems with tens of thousands of pieces.

11.3 Acknowledgements

We thank the editorial staff of the *Bild der Wissenschaft* and the publisher *Konradin Medien GmbH* for their permission to use the magazine and to make the dataset publicly available for research.

Appendix A

Supplementary Material

The following figures provide some experimental results that were omitted from the main text due to space limitations:

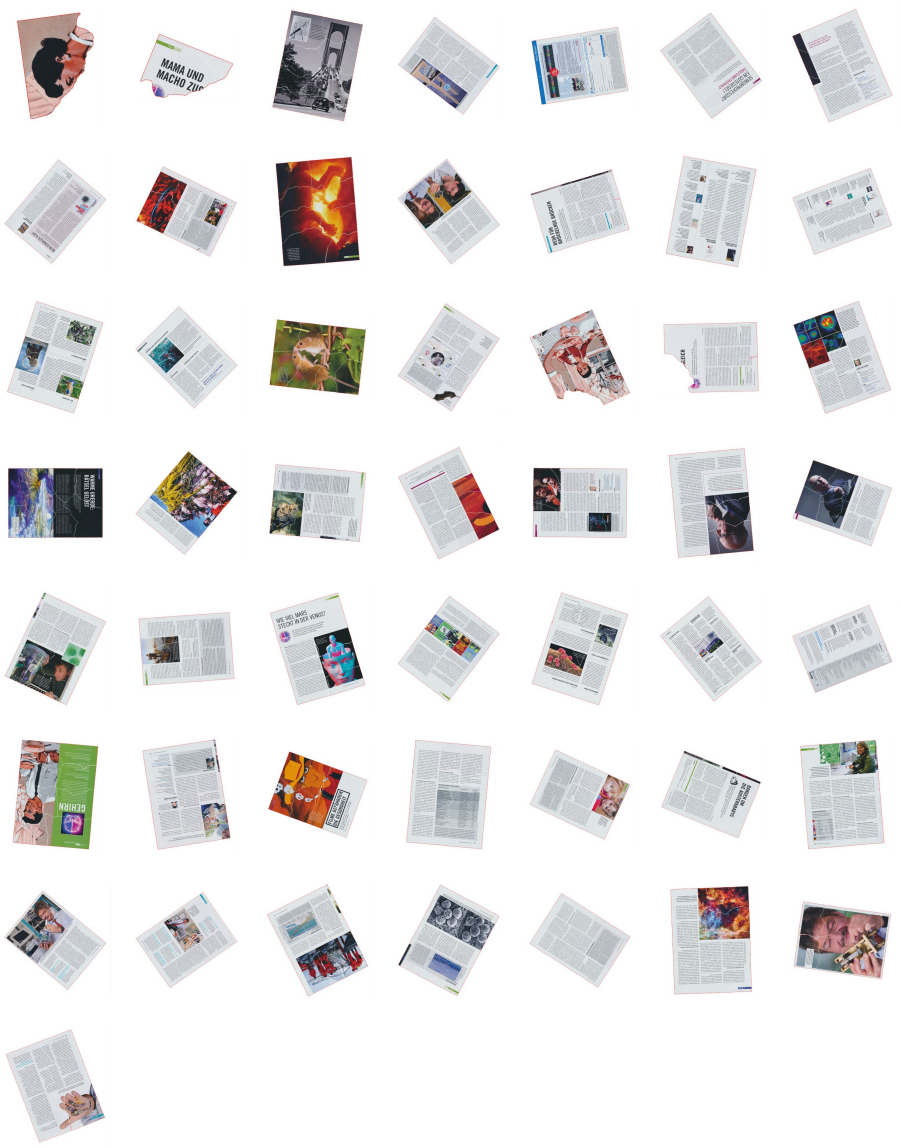


Figure A.1: Partial solutions obtained for the *bdw082010* dataset (48 pages, 8 pieces). We used our two-sided structural approach for the reconstruction ($K = 5$). For this experiment neither the number of pages nor the belonging of pieces to pages was assumed to be known (unrestricted reconstruction, see section 10.4).

Figure A.2: Partial solutions obtained for the *booklet* dataset (24 pages, 8 pieces). We used our two-sided structural approach for the reconstruction ($K = 5$). For this experiment neither the number of pages nor the belonging of pieces to pages was assumed to be known (unrestricted reconstruction, see section 10.4).

Figure A.3: Partial solutions obtained for the *stieber500p* dataset (60 pages, $\varnothing = 8.33$ pieces per page). We used our two-sided structural approach for the reconstruction ($K = 5$). For this experiment neither the number of pages nor the belonging of pieces to pages was assumed to be known (unrestricted reconstruction, see section 10.4).

Appendix B

Proofs and Remarks

Proof 1. We now prove that the expected number of hypotheses among all candidate sets amounts to

$$E\big[|Q|\big] = \eta n^3 p \sum_{k=0}^{\lceil \eta n \rceil - 1} (1-p)^k = \eta n^3 \left[1 - (1-p)^{\lceil \eta n \rceil}\right], \tag{B.1}$$

as stated in eq. (8.10) in chapter 8. To this end, recall that $0 < \eta \ll 1$ is the percentage of the piece's circumference c that determines the length of the L_1-neighborhood, i.e., $L_1 = \eta c$. Besides, n is the number of support points on both pieces, and p is the probability that a point within a L_1-neighborhood on the first piece is matched with a point within the L_1-neighborhood on the second piece. We assumed that points in both L_1-neighborhoods are distributed uniformly at random and independent of each other. Therefore, the probability for making a match was assessed as $p = 2L_2/L_1$, where L_2 is the size of the one-sided matching interval.

Under these premises we show by induction that the expected number of matches corresponds to a partial sum of a geometric series:

Base case: For the first point in the L_1-neighborhood on piece $\mathcal{P}_k$ we expect $a_0 = \eta n p$ matches, which is the first term in a geometric progression. Since each point from the L_1-neighborhood on piece $\mathcal{P}_l$ can contribute only a single match, there remain another $\eta n - \eta n p$ candidate points for matches with the next point on $\mathcal{P}_k$. Accordingly, we expect $a_1 = (\eta n - \eta n p)p = \eta n(1-p)p$ matches for the second point. The ratio of the two successive terms a_1 and a_0 differs by a constant factor $q = a_1/a_0 = 1 - p$.

Induction step: Let $k = \mathbb{N}_{>0}$ be given and suppose that $a_k/a_{k-1} = q = 1 - p$ holds for all $k < m$, with $a_k = \eta n(1-p)^k p$. Since any two subsequent terms differ only

by factor q, the first m terms form a geometric progression $a_0, a_0q, a_0q^2, \ldots, a_0q^{m-1}$. The sum of these terms, which corresponds to the m-th partial sum of the geometric series $s = \sum_{k=0}^{\infty} a_k$, gives us the number of expected matches for the first m points. Furthermore, for $s_m = \sum_{k=0}^{m-1} a_k$ it can be shown that $s_m = a_0 \frac{1-q^m}{1-q}$ holds:

$$s_m = \sum_{k=0}^{m-1} a_k = a_0 \sum_{k=0}^{m-1} q^k \tag{B.2}$$

$$\Leftrightarrow qs_m = a_0 \sum_{k=0}^{m-1} q^{k+1} \tag{B.3}$$

$$\Leftrightarrow s_m - qs_m = a_0(1 - q^m) \tag{B.4}$$

$$\Leftrightarrow s_m = a_0 \frac{1 - q^m}{1 - q} \tag{B.5}$$

Then, for the $(m + 1)$-th partial sum we have:

$$a_m = \left(\eta n - \sum_{k=0}^{m-1} a_k \right) p = (\eta n - s_m)\, p \tag{B.6}$$

$$= \left(\eta n - a_0 \frac{1 - q^m}{1 - q} \right) p \tag{B.7}$$

$$= \eta np - a_0(1 - (1 - p)^m) \tag{B.8}$$

$$= \eta n(1 - p)^m p \tag{B.9}$$

In the step from eq. (B.6) to (B.7) we make use of our inductive assumption. Again, $a_m/a_{m-1} = q$ holds, which means that the geometric progression was extended by one additional term. Consequently, we now have $s_{m+1} = \sum_{k=0}^{m} a_k = a_0 \frac{1-q^{m+1}}{1-q} = \eta n(1 - q^{m+1})$. Given that the L_1-neighborhood contains $m + 1$ points, we can approximate this value by $\lceil \eta n \rceil$. This yields the number of expected matches across the two pieces' L_1-neighborhoods:

$$\eta n \left[1 - (1 - p)^{\lceil \eta n \rceil} \right] \tag{B.10}$$

Multiplying this number with n^2 to account for all anchor points across the pieces yields the required quantity $E[|Q|]$. ■

Remark 1. To determine the maximal number of structured output predictions needed for an unrestricted reconstruction of M pages, one has two consider two aspects: First, a document graph needs to be constructed: We have to match each piece with every other piece exactly once, with the exception of registered counterparts. Let N be the

total number of pieces to be considered. There are $\frac{1}{2}(N^2 - N)$ unique piece-pairs, from which $\frac{N}{2}$ can be disregarded because counterparts must not be aligned with each other. For graph construction we thus require

$$\frac{1}{2}(N^2 - N) - \frac{N}{2} \tag{B.11}$$

comparisons of piece-pairs.

The second aspect ties to updates of this graph. Once after two (groups of) pieces and their counterparts have been merged, we must account for the new evidence and alter the graph's link structure and weights accordingly. After the first merging step, in which four pieces become combined into two new partial solutions, we need to update our structured output predictions for $2(N - 2k - 2)$ partial solutions, where $k = 1$. The reasoning is that due to the first merging step, we have effectively reduced the number of partial solutions to $N - 2k$, and each such new partial solution must be compared to all remaining solutions other than itself and its counterpart (hence the minus 2). The factor 2 in front is because both sides of the two new partial solutions must be considered separately in that process. Overall this update step is repeated at most $\frac{N-2}{2}$ times, which is why the maximum total number of updates corresponds to:

$$2 \sum_{k=1}^{\frac{N-2}{2}} (N - 2k - 2) \tag{B.12}$$

Together, the number of alignments required for (i) the graph construction and (ii) the repeated updates of the graph amounts to:

$$\frac{1}{2}(N^2 - N) - \frac{N}{2} + 2 \sum_{k=1}^{\frac{N-2}{2}} (N - 2k - 2) = N^2 - 4N + 4, \tag{B.13}$$

This follows directly from applying simple algebraic transformations. Note that in practice we sometimes encounter situations in which structured output prediction requires less computational effort; for instance, at later stages of reconstruction, some partial solutions already correspond to entire document pages, in which case MSAC typically outputs no hypotheses. Correspondingly, we also do not need to apply the stuctural SVM. Therefore, $N^2 - 4N + 4$ provides an upper bound on the number of piece-groups for which we have to apply *both*, our partial contour matching method (MSAC) *and* our structural SVM.

Appendix C

List of Publications

Parts of the work presented in this thesis have been published in the following conference papers and articles:

1. F. RICHTER, C. EGGERT, AND R. LIENHART. Fisher vector encoding of micro color features for (real world) jigsaw puzzles. In *IAPR 13th International Conference on Document Analysis and Recognition (ICDAR)*, 2015.

2. F. RICHTER, C. X. RIES, AND R. LIENHART. Evaluation of discriminative models for the reconstruction of hand-torn documents. In *Asian Conference on Computer Vision (ACCV)*, pages 671–686, 2014.

3. F. RICHTER, C. X. RIES, S. ROMBERG, AND R. LIENHART. Partial contour matching for document pieces with content-based prior. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2014.

4. F. RICHTER, C. X. RIES, N. CEBRON, AND R. LIENHART. Learning to reassemble shredded documents. *IEEE Transactions on Multimedia*, **15**(3):582–593, 2012.

5. F. RICHTER, C. X. RIES, AND R. LIENHART. A graph algorithmic framework for the assembly of shredded documents. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2011.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] The opencv library. http://opencv.org/.

[2] N. ALAJLAN. Solving square jigsaw puzzles using dynamic programming and the hungarian procedure. *American Journal of Applied Sciences*, **6**(11):1941–1947, 2009.

[3] F. AMIGONI, S. GAZZANI, AND S. PODICO. A method for reassembling fragments in image reconstruction. In *IEEE Int. Conf. on Image Processing (ICIP)*, pages III – 581–4 vol. 2, 2003.

[4] S. BELONGIE AND J. MALIK. Matching with shape contexts. In *IEEE Workshop on Content-based Access of Image and Video Libraries*, pages 20–26, 2000.

[5] A. BISWAS, P. BHOWMICK, AND B. B. BHATTACHARYA. Reconstruction of torn documents using contour maps. In *IEEE Int. Conf. on Image Processing (ICIP)*, pages III – 517–20, 2005.

[6] M. B. BLASCHKO AND C. H. LAMPERT. Learning to localize objects with structured output regression. In *10th European Conf. on Computer Vision: Part I (ECCV)*, pages 2–15, 2008.

[7] H. BUNKE AND U. BÜHLER. Applications of approximate string matching to 2d shape recognition. *Pattern Recognition*, **26**(12):1797–1812, 1993.

[8] S. CAO, H. LIU, AND S. YAN. Automated assembly of shredded pieces from multiple photos. In *IEEE Int. Conf. on Multimedia and Expo (ICME)*, pages 358–363, 2010.

[9] O. CHAPELLE, C. B. DO, Q. V. LE, A. J. SMOLA, AND C. H. TEO. Tighter bounds for structured estimation. In *Conf. on Neural Information Processing Systems (NIPS)*, 2008.

[10] L. CHEN, R. FERIS, AND M. TURK. Efficient partial shape matching using smith-waterman algorithm. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2008.

[11] T. S. CHO, S. AVIDAN, AND W. T. FREEMAN. A probabilistic image jigsaw puzzle solver. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 183–190, 2010.

[12] H. C. DA GAMA LEITÃO AND J. STOLFI. A multiscale method for the reassembly of two-dimensional fragmented objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **24**:1239–1251, 2002.

[13] E. D. DEMAINE AND M. L. DEMAINE. Jigsaw puzzles, edge matching, and polyomino packing: Connections and complexity. *Graphs and Combinatorics*, **23**(1):195–208, 2007.

[14] D. DOUGLAS AND T. PEUCKER. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, **10**(2):112–122, 1973.

[15] P. F. FELZENSZWALB, R. B. GIRSHICK, D. MCALLESTER, AND D. RAMANAN. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **32**(9):1627–1645, 2010.

[16] P. F. FELZENSZWALB AND D. P. HUTTENLOCHER. Distance transforms of sampled functions. *Theory of Computing*, **8**(19):415–428, 2012.

[17] M. FISCHLER AND R. BOLLES. Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography. *Communications of the ACM*, **24**(12):381–395, 1981.

[18] H. FREEMAN AND L. GARDER. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Transactions on Electronic Computers*, **13**(2):118–127, 1964.

[19] Y. FREUND. Boosting a weak learning algorithm by majority. *Information and Computation*, **121**(2):256–285, 1995.

[20] Y. FREUND AND R. E. SCHAPIRE. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, **55**(1):119–139, 1997.

[21] Y. FREUND AND R. E. SCHAPIRE. A short introduction to boosting. In *16th Int. Joint Conf. on Artificial Intelligence*, pages 1401–1406, 1999.

[22] J. FRIEDMAN, T. HASTIE, AND R. TIBSHIRANI. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, **28**(2):337–374, 2000.

[23] A. GALLAGHER. Jigsaw puzzles with pieces of unknown orientation. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 382–389, 2012.

[24] T. GELLER. Darpa shredder challenge solved. *Communications of the ACM*, **55**(8):16–17, 2012.

[25] D. GOLDBERG, C. MALON, AND M. BERN. A global approach to automatic solution of jigsaw puzzles. *Computational Geometry*, **28**:165–174, 2004.

[26] R. M. HARALICK, K. SHANMUGAM, AND I. DINSTEIN. Textural features for image classification. In *IEEE Transactions on Systems, Man, and Cybernetics*, pages 610–621, 1973.

[27] J. HAUBRICH. *Compendium of Card Matching Puzzles*. Self-published, 1995.

[28] M. J. HEULE. Solving edge-matching problems with satisfiability solvers. In *Int. Workshop on Logic and Search*, pages 88–102. University of Leuven, 2008.

[29] C. HOLLITT AND A. S. DEEB. Determining image orientation using the hough and fourier transforms. In *27th Conf. on Image and Vision Computing New Zealand (IVCNZ)*, pages 346–351, 2012.

[30] T. JAAKKOLA AND D. HAUSSLER. Exploiting generative models in discriminative classifiers. In *Conf. on Advances in Neural Information Processing Systems (NIPS)*, pages 487–493. MIT Press, 1998.

[31] T. JOACHIMS. Making large-scale SVM learning practical. In B. SCHÖLKOPF, C. BURGES, AND A. SMOLA, editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.

[32] T. JOACHIMS, T. FINLEY, AND C.-N. J. YU. Cutting-plane training of structural svms. *Machine Learning*, **77**(1):27–59, 2009.

[33] E. JUSTINO, L. S. OLIVEIRA, AND C. FREITAS. Reconstructing shredded documents through feature matching. *Forensic Science Int.*, **160**(2–3):140–147, 2006.

[34] W. KONG AND B. B. KIMIA. On solving 2d and 3d puzzles using curve matching. In *IEEE Con. on Computer Vision and Pattern Recognition (CVPR)*, pages II–583 – II–590 vol. 2, 2001.

[35] J. KRUSKAL. On the shortest spanning subtree and the traveling salesman problem. *Proceedings of the American Mathematical Society*, **7**(1):48–50, 1956.

[36] J. C. MCBRIDE AND B. B. KIMIA. Archaeological fragment reconstruction using curve-matching. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshop (CVPRW)*, 2003.

[37] V. NOVK. Cyclically ordered sets. *Czechoslovak Mathematical Journal*, **32**(3):460–473, 1982.

[38] C. PAPAODYSSEUS, T. PANAGOPOULOS, M. EXARHOS, C. TRIANTAFILLOU, D. FRAGOULIS, AND C. DOUMAS. Contour-shape based reconstruction of fragmented, 1600 bc wall paintings. *IEEE Transactions on Signal Processing*, **50**:1277–1288, 2002.

[39] B. PEPIK, M. STARK, P. GEHLER, AND B. SCHIELE. Occlusion patterns for object class detection. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[40] D. POMERANZ, M. SHEMESH, AND O. BEN-SHAHAR. A fully automated greedy square jigsaw puzzle solver. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 9–16, 2011.

[41] F. RICHTER, C. EGGERT, AND R. LIENHART. Fisher vector encoding of micro color features for (real world) jigsaw puzzles. In *IAPR 13th Int. Conf. on Document Analysis and Recognition (ICDAR)*, 2015.

[42] F. RICHTER, C. X. RIES, N. CEBRON, AND R. LIENHART. Learning to reassemble shredded documents. *IEEE Trans. on Multimedia*, **15**(3):582–593, 2012.

[43] F. RICHTER, C. X. RIES, AND R. LIENHART. A graph algorithmic framework for the assembly of shredded documents. In *IEEE Int. Conf. on Multimedia and Expo (ICME)*, 2011.

[44] F. RICHTER, C. X. RIES, AND R. LIENHART. Evaluation of discriminative models for the reconstruction of hand-torn documents. In *Asian Conf. on Computer Vision (ACCV)*, pages 671–686, 2014.

[45] F. RICHTER, C. X. RIES, S. ROMBERG, AND R. LIENHART. Partial contour matching for document pieces with content-based prior. In *IEEE Int. Conf. on Multimedia and Expo (ICME)*, 2014.

[46] J. SANCHEZ, F. PERRONNIN, T. MENSINK, AND J. VERBEEK. Image classification with the fisher vector: Theory and practice. *Int. Journal of Computer Vision*, **105**(3):222–245, 2013.

[47] R. E. SCHAPIRE. The strength of weak learnability. *Machine Learning*, **5**(2):197–227, 1990.

[48] D. SHOLOMON, O. DAVID, AND N. S. NETANYAHU. A genetic algorithm-based solver for very large jigsaw puzzles. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1767–1774, 2013.

[49] P. D. SMET. Semi-automatic forensic reconstruction of ripped-up documents. In *IAPR 10th Int. Conf. on Document Analysis and Recognition (ICDAR)*, pages 703–707, 2009.

[50] T. F. SMITH AND M. S. WATERMAN. Identification of common molecular subsequences. *Journal of Molecular Biology*, **147**(1):195–197, 1981.

[51] A. STIEBER, J. SCHNEIDER, B. NICKOLAY, AND J. KRÜGER. A contour matching algorithm to reconstruct ruptured documents. In *DAGM Conf. on Pattern recognition*, 2010.

[52] S. SUZUKI AND K. ABE. Topological structural analysis of digital binary images by border following. *Computer Vision, Graphics, and Image Processing (CVGIP)*, **30**(1):32–46, 1985.

[53] P. H. S. TORR AND A. ZISSERMAN. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, **78**(1):138–156, 2000.

[54] I. TSOCHANTARIDIS, T. JOACHIMS, T. HOFMANN, AND Y. ALTUN. Large margin methods for structured and interdependent output variables. *The Journal of Machine Learning Research*, **6**:1453–1484, 2005.

[55] L. G. VALIANT. A theory of the learnable. *Communications of the ACM*, **27**(11):1134–1142, 1984.

[56] V. N. VAPNIK. *The nature of statistical learning theory*. Springer-Verlag, 1995.

[57] V. N. VAPNIK. *Statistical learning theory*. Wiley, 1998.

[58] A. VEDALDI AND B. FULKERSON. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[59] P. VIOLA AND M. J. JONES. Robust real-time face detection. *Int. Journal of Computer Vision*, **57**(2):137–154, 2004.

[60] R. W. WEBSTER, P. S. LAFOLLETTE, AND R. L. STAFFORD. Isthmus critical points for solving jigsaw puzzles in computer vision. *IEEE Transactions on Systems, Man and Cybernetics*, **21**(5):1271–1278, 1991.

[61] H. J. WOLFSON. On curve matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **12**(5):483–489, 1990.

[62] X. YANG, N. ADLURU, AND L. J. LATECKI. Particle filter with state permutations for solving image jigsaw puzzles. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2873–2880, 2011.

[63] Y. YANG AND D. RAMANAN. Articulated pose estimation with flexible mixtures-of-parts. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1385–1392, 2011.

[64] F. YAO AND G. SHAO. A shape and image merging technique to solve jigsaw puzzles. *Pattern Recognition Letters*, **24**(12):1819–1835, 2003.

[65] C.-N. J. YU, T. JOACHIMS, R. ELBER, AND J. PILLARDY. Support vector training of protein alignment models. *Journal of Computational Biology*, **15**(7):867–880, 2008.

[66] L. ZHU, T. ZHOU, AND D. HU. Globally consistent reconstruction of ripped-up documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, **30**(1):1–13, 2008.

[67] L. ZHU, Z. ZHOU, J. ZHANG, AND D. HU. A partial curve matching method for automatic reassembly of 2d fragments. In *Int. Conf. on Intelligent Computing (ICIC)*, pages 645–650, 2006.